InfiniBand[TM] Architecture Specification Volume 1

Release 1.0.a

June 19, 2001
Final

Table 1 Revision History

Revision	Release Date	
1.0	9/26/2000	Release 1.0
1.0.a	6/19/2001	Release 1.0 augmented with errata material. Updates only correct errors - no additional features have been added.

LEGAL DISCLAIMER

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1: Introduction

This is Volume 1 of the InfiniBand™ Architecture specification. It is the first in a series of documents that describe the architecture.

1.1 Acknowledgments

The following persons were instrumental in the development of this volume of the InfiniBand™ Architecture specification:

Steering Committee Members

Co-chairs:

Tom Bradicich	Tom Macdonald

Members:

Jacqueline Balfour	Ed Miller	Bob Zak
Balint Fleischer	John Pescatore	Dr. Alfred Hartmann
David Heisey	Jim Pinkerton	
Ken Jansen	Martin Whittaker	

Technical Working Group Members

Co-chairs:

Dwight Barron	Irv Robinson	David Wooten

Members:

Dr. Alan Benner	Dr. Alfred Hartmann	Ed Miller
Mark Bradley	Michael Krause	Greg Still
Wolfgang Christl	Bill Lynn	Ken Ward

Working Group Co-Chairs

Link Working Group (LWG):

Daniel Cassiday	Michael Krause

Software Working Group (SWG):

Ed Miller	Renato J. Recio	Jim Pinkerton

Management Working Group (MgtWG):

David W. Abmayr	Brian Forbes	William H. Swortwood

Contributors

David W. Abmayr
Dr. Ramon Acosta
Dr. Alan Benner
Frank L. Berry
Bruce Beukema
Suri Brahmaroutu
David M. Brean
Daniel Cassiday
Ian Colloff
Joe Cowan
Olivier Crémel
Diego Crupnicoff
Paul Culley
Roger Cummings
George Dake
Ellen Deleganes
Kevin Deierling
Scott Feller
Brian Forbes
Dan Fowler
Giles Frazier
Bill Futral
Narayanan Ganapathy
David Garcia
Nancy J. Golio
Paul Grun
James Hamrick
Ariel Hendel
Jeff Hilland
Jenwei Hsieh
Christopher J. Jackson
Jeff Jilg
Dave Kasberg
Dr. Ted Kim
Dr. Hiro Kishimoto
Michael Krause
Alan Langerman
James W. Livingston
Venitha L. Manter
Dr. Pankaj Mehra
Charles Monia
Mark Myers
Neil MacLean
Tarl Neustaedter
Shravan Pargal
Joe Pelissier
Dr. Gregory F. Pfister
Jim Pinkerton
Vandana Rao
Renato J. Recio
Tom Ryle
Hide Senta
Michael Shinkarovsky
Cris Simpson
William H. Swortwood
Chris Szeto
Pat Thaler
Franco Travastino
Dono Van-Mierop
Ken Ward
Kurt Ware
Tom Webber
Jeff Young
Dr. Mazin Yousif

1.2 INFINIBAND CONCEPTUAL OVERVIEW

The InfiniBand™ Architecture Specification describes a first order interconnect technology for interconnecting processor nodes and I/O nodes to form a system area network. The architecture is independent of the host operating system (OS) and processor platform.

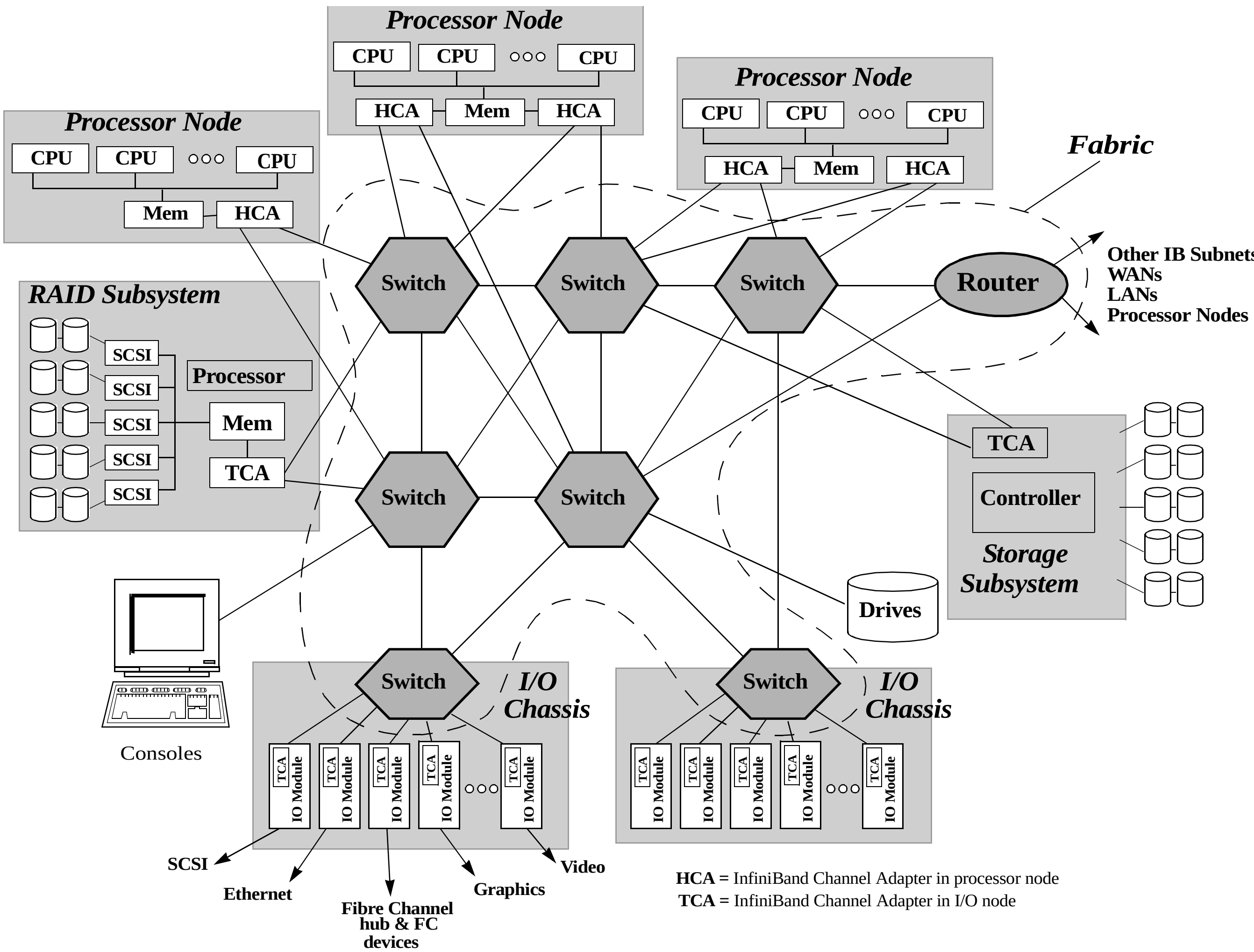


Figure 1 IBA System Area Network

1.2.1 THE PROBLEM

Existing interconnect technologies have failed to keep pace with computer evolution and the increased burden imposed on data servers, application processing, and enterprise computing created by the popular success of the internet. High-end computing concepts such as clustering, fail-safe, and 24x7 availability demand greater capacity to move data between processing nodes as well as between a processor node and I/O devices. These trends require higher bandwidth and lower latencies, they are pushing more functionality down to the I/O device, and they are demanding greater protection, higher isolation, deterministic behavior, and a higher quality of service than currently available.

1.2.2 FEATURES

InfiniBand™ Architecture (IBA) is designed around a point-to-point, switched I/O fabric, whereby end node devices (which can range from very inexpensive I/O devices like single chip SCSI or ethernet adapters to very complex host computers) are interconnected by cascaded switch devices. The physical properties of the IBA interconnect support two predominant environments, with bandwidth, distance and cost optimizations appropriate for these environments:

- Module-to-module, as typified by computer systems that support I/O module add-in slots
- Chassis-to-chassis, as typified by interconnecting computers, external storage systems, and external LAN/WAN access devices (such as switches, hubs, and routers) in a data-center environment.

The IBA switched fabric provides a reliable transport mechanism where messages are enqueued for delivery between end nodes. In general, message content and meaning is not specified by InfiniBand Architecture, but rather is left to the designers of end node devices and the processes that are hosted on end node devices. IBA defines hardware transport protocols sufficient to support both reliable messaging (send/receive) and memory manipulation semantics (e.g. remote DMA) without software intervention in the data movement path. IBA defines protection and error detection mechanisms that permit IBA transactions to originate and terminate from either privileged kernel mode (to support legacy I/O and communication needs) or user space (to support emerging interprocess communication demands).

The IBA Specification also addresses the need for a rich manageability infrastructure to support interoperability between multiple generations of IBA components from many vendors over time. This infrastructure provides ease of use and consistent behavior for high volume, cost sensitive deployment environments. IBA also specifies interfaces for industry standard management that interoperate with enterprise class management tools for configuration, asset management, error reporting, performance metric collection, and topology management necessary for data center deployment of IBA.

1.2.3 BENEFITS

For all of the revolutionary aspects of IBA, the architecture has been carefully designed to minimize disruption of prevailing market paradigms and business practices. By simultaneously supporting board and chassis interconnections, it is expected that vendors are able to adopt InfiniBand Architecture technology for use in future generations of existing products, within current business practices, to best support their customers needs.

IBA can support bandwidths that are anticipated to remain an order of magnitude greater than prevailing I/O media (SCSI, Fibre Channel, Ethernet). This ensures its role as the common interconnect for attaching I/O media using these technologies. Reinforcing this point is IBA's native use of IPv6 headers, which supports extremely efficient junctions between IBA fabrics and traditional internet and intranet infrastructures.

IBA supports implementations as simple as a single computer system, and can be expanded to include: replication of components for increased system reliability, cascaded switched fabric components, additional I/O units for scalable I/O capacity and performance, additional host node computing elements for scalable computing, or any combinations thereof. InfiniBand Architecture is a revolutionary architecture that enables computer systems to keep up with the ever increasing customer requirement for increased scalability, increased bandwidth, decreased CPU utilization, high availability, high isolation, and support for Internet technology.

Being designed as a first order network, IBA focuses on moving data in and out of a node's memory and is optimized for separate control and memory interfaces. This permits hardware to be closely coupled or even integrated with the node's memory complex, removing any performance barriers. IBA is flexible enough to be implemented as a second order network permitting legacy and migration. Even when implemented as a second order network, IBA's memory optimization operation permits maximum available bandwidth utilization and increases CPU efficiency.

1.3 SCOPE

IBA supports a range of applications from being the backplane interconnect of a single host, to a complex system area network consisting of multiple independent and clustered hosts and I/O components.

For the single host environments, as depicted in Figure 2, each IBA fabric serves as a private I/O interconnect for its host and provides connectivity between the host's CPU/memory complex and a number of I/O modules. For this environment, all devices are dedicated to the host.

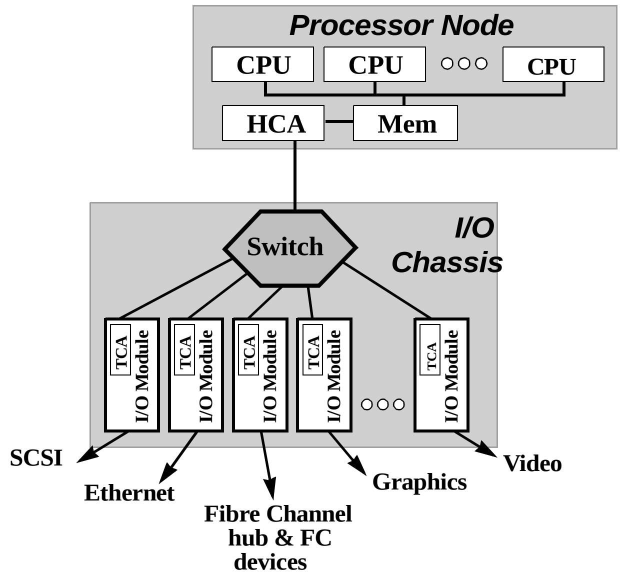


Figure 2 Single Host Environment

On the other end of the scale is multiple host connectivity as depicted in Figure 1. Here a single fabric or even multiple fabrics interconnect numerous hosts and various I/O units. Some hosts might share I/O devices and others do not. Interprocess communication between hosts becomes a very significant objective. Trivial fabric management is no longer sufficient as network administrators desire additional features to maintain separation and assure deterministic behavior.

The architecture not only specifies the mechanisms for I/O and interprocess communication, but it also specifies an extensive set of management mechanisms that are flexible enough to permit single host environments with out undue burden and costly fabric managers and at the same time support very complex system area networks (SAN) and feature rich fabric management.

1.4 DOCUMENT ORGANIZATION

1.4.1 SERIES OF VOLUMES

There are two volumes that comprise the InfiniBand normative specifications suite:

Volume 1 - specifies the core InfiniBandTM Architecture. It provides normative information required for IBA operation for switches, routers, host channel adapters for processor nodes, target channel adapters for I/O devices, and management.

Volume 2 - specifies electrical & mechanical configurations. It specifies requirements for a number of different physical media and signaling rates, defines mechanical form factors, and specifies physical and chassis management requirements.

1.4.2 Volume 1 Organization

1.5 Document Conventions

1.5.1 Byte Ordering

This specification uses Big Endian byte ordering. For fields greater than one byte in size this means that the most significant byte of each field is transmitted first as illustrated in Figure 3.

Figure 3 Byte Order for Multiple Byte Fields

Byte offset	previous field
+0	**Most Significant Byte**
+1	**Second Most Significant Byte**
. . .	**o** **o** **o**
+n	**Least Significant Byte**

Unless specifically stated otherwise, the text of this document lists fields in the order of transmission. In most cases, multiple byte fields are aligned to start or end on a 32-bit boundary. For clarity, certain figures show fields ordered in 32 bit words. These words are in big endian format and implementations targeted for little endian processing need to pay particular attention to byte ordering to assure correct operation since little endian processing tends to place the least significant bytes in lower byte offsets.

Figure 4 illustrates how numeric and bit significant fields should be interpreted.

Figure 4 Byte Order Examples

bits / Byte Offset	b7 … b0 Byte 0,4,8.	b7 … b0 Byte 1,5,9,...	b7 … b0 Byte 2,6,10,...	b7 … b0 Byte 3,7,11,...
0-3	b15 **16-bit field** b0		b15 **16-bit field** b0	
4-7	b31 **32-bit field** b0			
8-11	b7 **1-byte** b0	b23 **24-bit field** b0		
12-15	b23 **24-bit field** b0			b7 **1-byte** b0
16-19	b47 **48-bit field (high)** b16			
20-23	b15 **48-bit field (low)** b0		b47 **48-bit field (high bytes)** b32	
24-27	b31 **48-bit field (low bytes)** b0			
28-31	b63 **64-bit field (high bytes)** b32			
32-35	b31 **64-bit field (low bytes)** b0			
36-39	b127 **128-bit field (highest bytes)** b96			
40-43	b95 … b64			
44-47	b63 … b32			
48-51	b31 **128-bit field (lowest bytes)** b0			

Bit fields with other than byte granularity follow the same rules - that is, the most significant bits of the field occupies the higher order bits of the lowest byte offset with least significant bits being in the lowest byte offset as illustrated in Figure 5.

Figure 5 Bit Order Examples

Previous Byte	First Byte								Next Byte								Following Byte
	5-bit field					**3-bit field**			**2-bit**		**6-bit field**						
	b4	b3	b2	b1	b0	b2	b1	b0	b1	b0	b5	b4	b3	b2	b1	b0	
	4-bit field				**12-bit field**												
	b3	b2	b1	b0	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	
	14-bit field														**2-bit**		
	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0	b1	b0	

1.5.2 NUMERIC VALUES

Unless otherwise stated numerical values without qualifiers are decimal. This document uses the following qualifiers:

- 0x prefixed to a hexidecimal value (e.g., 0x15F7)
- b' prefixed to a binary value (e.g., b'0110)

An obvious exception are binary numbers used in figures and tables

In table headings a colon is used to specify a range of bits (e.g. Bits 7:0) and table values in that column are binary numbers.

A dash between two numbers represents a range (e.g. 0-3 = zero to three)

Global IDs are 128-bit values specified in the format :
value:value:value:value:value:value:value:value
Where each value represents a 4-digit hexidecimal number (e.g., FF02:0:0:0:0:0:0:1)

1.6 DISCLAIMER

Like any document, this specification is subject to errata for correctness, clarity, and enhancements. The InfiniBand℠ Trade Association hosts a web site at http://www.InfiniBandTA.org. Please visit this site to check for errata and updates to this specification.

CHAPTER 2: GLOSSARY

Active	Describes an entity initiating a communication establishment request (e.g., TCP CONNECT).
Address Handle	An object that contains the information necessary to transmit messages to a remote port over Unreliable Datagram service.
Address Vector	A collection of address and Path information specifying a remote port and the parameters to be used when communicating with it.
AETH	Ack Extended Transport Header
AM	Attribute Modifier.
Asynchronous error	A permanent error that cannot be reported through immediate or completion error handling mechanisms at the local end. Asynchronous errors may be unaffiliated or may be affiliated with a specific Completion Queue, End to End Context, or Queue Pair.
Attribute	The collection of management data carried in a Management Datagram.
Automatic Path Migration	The process in which a Channel Adapter, on a per-Queue Pair basis, signals another CA to cause Path Migration to a preset alternate Path. Automatic Path Migration uses a bit in a request or response packet (MigReq) to signal the other channel adapter to migrate to the predefined alternate path.
B_Key	See Baseboard Management Key.
Base LID	The numerically lowest Local Identifier that refers to a Port. The Path Bits of a Base LID are always zero.
Baseboard Managed Unit	Any Unit which provides InfiniBand™ specification defined information about itself by a Baseboard method MAD operation through the InfiniBand™ link.
Baseboard Management Key	A construct that is contained in IBA management datagrams to authenticate that the sender is allowed to perform the requested operation.
Binding	The act of associating a virtual address range in a specified Memory Registration with a Memory Window.
BTH	Base Transport Header.

CA See Channel Adapter.

Channel The association of two queue pairs for communication.

Channel Adapter Device that terminates a link and executes transport-level functions. One of Host Channel Adapter or Target Channel Adapter.

Channel Interface The presentation of the channel to the Verbs Consumer as implemented through the combination of the Host Channel Adapter, associated firmware, and device driver software.

Channel, Reliable Datagram See Reliable Datagram Channel.

CI See Channel Interface.

Client The active entity in an active/passive communication establishment exchange.

CM See Communication Manager.

CME Chassis Management Entity.

Communication Manager The software, hardware, or combination of the two that supports the communication management mechanisms and protocols.

Completion Error Permanent interface or processing error reported through completion status.

Completion Queue A queue containing one or more Completion Queue Entries. Completion Queues are internal to the Channel Interface, and are not visible to verb consumers.

Completion Queue Entry The Channel Interface-internal representation of a Work Completion.

Connection An association between a pair of entities (e.g., processes) over one or more Channels.

Consumer See Verbs Consumer.

CQE Completion Queue Entry, commonly pronounced "cookie".

CRC Cyclic Redundancy Check.

Data Payload The data, not including any control or header information, carried in one packet.

Data Segment A tuple in a Work Request that specifies a virtually contiguous buffer for Host Channel Adapter access. Each Data Segment consists of a Virtual

Address, an associated Local Key or Remote Key, and a length.

DETH Datagram Extended Transport Header.

DGID Destination Globally Unique Identifier.

DLID Destination Local Identifier

EEC See End to End Context.

EECN See End to End Context Number.

EE Context See End to End Context.

End to End Context The endpoint of a Reliable Datagram channel.

End to End Context Number Identifies a specific End to End Context within a Channel Adapter.

End to End Flow Control A mechanism to prevent a sender from transmitting messages during periods when receive buffers are not posted at the recipient.

Fabric The collection of Links, Switches, and Routers that connects a set of Channel Adapters.

Gb/s Giga-bits per second (10^9 bits per second)

GB/s Giga-bytes per second (10^9 bytes per second)

General Service Interface An interface providing management services (e.g., connection, performance, diagnostics) other than subnet management.

GID See Global Identifier.

Global Identifier A 128-bit identifier used to identify a port on a channel adapter, a port on a router, or a multicast group. GIDs are valid 128-bit IPv6 addresses (per RFC 2373) with additional properties / restrictions defined within IBA to facilitate efficient discovery, communication, and routing.

Global Route Header Routing header present in InfiniBand™ Architecture packets targeted to destinations outside the sender's local subnet.

Globally Unique Identifier A number that uniquely identifies a device or component.

GMP General Management Packet.

GRH See Global Route Header.

GSI See General Service Interface.

GUID See Globally Unique Identifier.

HCA See Host Channel Adapter.

Host One or more Host Channel Adapters governed by a single memory/CPU complex.

Host Channel Adapter A Channel Adapter that supports the Verbs interface.

IBA InfiniBand™ Architecture.

IB-ML InfiniBand™ Management Link.

ICRC See Invariant CRC.

Immediate Data Data contained in a Work Queue Element that is sent along with the payload to the remote Channel Adapter and placed in a Receive Work Completion.

Immediate Error A permanent Interface Error reported through the verb status.

Initiator The source of requests.

Interface Error An error due to an invalid field in a Work Request.

Invalid Key See Key.

Invariant CRC A CRC covering the fields in a packet that do not change from the source to the destination.

I/O Input/Output.

I/O Controller One of the two architectural divisions of an I/O Unit. An I/O controller (IOC) provides I/O services, while a Target Channel Adapter provides transport services.

I/O Unit An I/O unit (IOU) provides I/O service(s). An I/O unit consists of one or more I/O Controllers attached to the fabric through a single Target Channel Adapter.

I/O Virtual Address An address having no direct meaning to the Host processor, intended for use only in describing a Local or Remote memory buffer to the Host Channel Adapter.

IOC See I/O Controller.

IOU See I/O Unit.

IPv6 — Internet Protocol, version 6

IPv6 Address — A 128-bit address constructed in accordance with IETF RFC 2460 for IPv6.

Key — A construct used to limit access to one or more resources, similar to a password. The following keys are defined by the InfiniBand™ Architecture:

- Baseboard Management Key
- Local Key
- Management Key
- Queue Key
- Partition Key
- Remote Key

L_Key — See Local Key.

LID — See Local Identifier.

LID Mask Control — A per-port value assigned by the Subnet Manager. The value of the LMC specifies the number of Path Bits in the Local Identifier.

Link — A full duplex transmission path between any two network fabric elements, such as Channel Adapters or Switches.

LMC — See LID Mask Control.

Local Identifier — An address assigned to a port by the Subnet Manager, unique within the subnet, used for directing packets within the subnet. The Source and Destination LIDs are present in the Local Route Header. A Local Identifier is formed by the sum of the Base LID and the value of the Path Bits.

Local Key — An opaque object, created by a verb, referring to a Memory Registration, used with a Virtual Address to describe authorization for the HCA hardware to access local memory. It may also be used by the HCA hardware to identify the appropriate page tables for use in translating virtual to physical addresses.

Local Route Header — Routing header present in all InfiniBand™ Architecture packets, used for routing through switches within a subnet.

Local Subnet — The collection of links and Switches that connect the Channel Adapters of a particular subnet.

LRH — See Local Route Header.

M_Key See Management Key.

MAD See Management Datagram.

Managed Unit A Unit which provides Vital Product Data about itself to an external entity, and is managed by that entity.

Management Datagram Refers to the contents of an Unreliable Datagram packet used for communication among HCAs, switches, routers, and TCAs to manage the fabric. InfiniBand™ Architecture describes the format of a number of these management commands.

Management Key A construct that is contained in IBA management datagrams to authenticate the sender to the receiver.

Maximum Transfer Unit See Path Maximum Transfer Unit.

MB/s Mega-bytes per second (10^6 bytes per second)

Memory Protection Attributes The access rights granted to Memory Registrations.

Memory Region A virtually contiguous area of arbitrary size within a Consumer's address space that has been registered, enabling HCA local access and optional remote access.

Memory Region Handle An opaque object returned to the consumer when the consumer registers a Memory Registration. The Memory Region Handle is used to specify the registered region to the memory management verbs.

Memory Registration The act of registering a host Memory Registration for use by a consumer. The memory registration operation returns a Memory Region Handle. The process provides this with any reference to a virtual address within the memory region.

Memory Window An allocated resource that enables remote access after being bound to a specified area within an existing Memory Registration. Each Memory Window has an associated Window Handle, set of access privileges, and current R_Key.

Message A transfer of information between two or more Channel Adapters that consists of one or more packets.

Message-Level Flow Control See End to End Flow Control.

Message Sequence Number — A value returned as part of an acknowledgement by the responder to the requestor, indicating the last message completed. Contrast Packet Sequence Number.

Modifiers — In a verb definition, the list of input and output objects that specify how, and on what, the verb is to be executed.

MSN — See Message Sequence Number.

MTU — Maximum Transfer Unit, see Path Maximum Transfer Unit.

Multicast — A facility by which a packet sent to a single address may be delivered to multiple ports.

Multicast Identifier — An identifier for a set of ports making up a Multicast Group, typically belonging to different Channel Adapters. On a subnet, Multicast Identifiers share the address space of Local Identifiers.

Multicast Group — A collection of Channel Adapter ports that receive Multicast packets sent to a single address.

NQ — Notification Queue.

Out-of-band Management — Management messages which traverse a transport other than the InfiniBand™ fabric.

Outstanding —

1) The state of a Work Request after it has been posted on a Work Queue, but before the retrieval of the Work Completion by the consumer.
2) The state of a packet that has been sent onto the fabric but has not been acknowledged.

P_Key — See Partition Key.

Packet — The indivisible unit of IBA data transfer and routing, consisting of one or more headers, a Packet Payload, and one or two CRCs.

Packet Payload — The portion of a Packet between (not including) any Transport header(s) and the CRCs at the end of each packet. The packet payload contains up to 4096 bytes.

Packet Sequence Number — A value carried in the Base Transport Header that allows the detection and re-sending of lost packets.

Partition — A collection of Channel Adapter ports that are allowed to communicate with one another. Ports may be members of multiple partitions simultaneously. Ports in different partitions are unaware of each other's presence insofar as possible.

Partition Key A value carried in packets and stored in Channel Adapters that is used to determine membership in a partition.

Default Partition Key: A partition key special value providing Full membership in the default partition. See Partition Membership Type.

Invalid Partition Key: A special value that indicates that the Partition Key Table entry does not contain a valid key.

Partition Key Table A table of partition keys present in each Port.

Partition Key Table Index (P_Key_ix) An index into the partition key table.

Partition Manager The entity that manages partition keys and membership.

Partition Membership Type The high-order bit of the partition key is used to record the type of membership in an Port's partition table: 0 for Limited, 1 for Full. Limited members cannot accept information from other Limited members, but communication is allowed between every other combination of membership types.

Passive Describes an entity waiting to receive a communication establishment request (e.g., TCP LISTEN).

Path The collection of links, switches, and routers a message traverses from a source Channel Adapter to a destination channel adapter. Within a subnet, a path is defined by the tuple <SLID, DLID, SL>.

Path Bits The portion of a Local Identifier that may be changed to vary the Path through the subnet to a particular Port. If the Path Bits are zero, the Local Identifier is equal to the Base LID. The number of Path Bits applicable to a particular port is specified by the Subnet Manager through the LID Mask Control value.

Path Maximum Transfer Unit The maximum size of the Packet Payload supported along a Path from source to destination. PMTU is described in terms of the payload size, and may be 256, 512, 1024, 2048, or 4096 bytes.

Path Migration The modification of the Path used by a connection.

PD See Protection Domain.

Peer

1) One of the agents in an active/active connection establishment exchange.
2) A generic term for the entity at the other end of a connection.

Pinning memory A function supplied by the OS which forces the memory region to be resident and keeps the virtual-to-physical translations constant from the HCA point of view.

PM See Partition Manager.

PMTU See Path Maximum Transfer Unit.

Port Location on a Channel Adapter or Switch to which a link connects. There may be multiple ports on a single Channel Adapter, each with different context information that must be maintained. Switches/switch elements contain more than one port by definition.

Post To place a Work Request on a Work Queue.

Private Data A field present in Communication Management messages that is opaque at all IBA layers. Consumers may use this field to "piggy-back" additional information over the CM message exchange.

Processing Error A processing error is an error that occurs when the Host Channel Adapter is performing the unit of work described by the Work Queue Element and is unable to complete the request successfully due to an error that is returned by the transport protocol.

Protection Domain A mechanism for associating Queue Pairs, Address Handles, Memory Windows, and Memory Registrations.

PSN See Packet Sequence Number.

Q_Key See Queue Key.

QoS See Quality of Service.

QP See Queue Pair.

Quality of Service Metrics that predict the behavior, reliability, speed, and latency of a given network connection.

Queue Key A construct that is used to validate a remote sender's right to access a local Receive Queue for the Unreliable Datagram and Reliable Datagram service types. If the Q_Key present in an incoming packet does not match the value stored in the receiving QP, the packet shall be dropped.

Queue Pair Consists of a Send Work Queue and a Receive Work Queue. Send and receive queues are always created as a pair and remain that way throughout their lifetime. A Queue Pair is identified by its Queue Pair Number.

Queue Pair Context The information that pertains to a particular Queue Pair, such as the current Work Queue Elements, Packet Sequence Numbers, transmission parameters, etc.

Queue Pair Handle An opaque object that refers to a specific Queue Pair. A Queue Pair Handle is returned by the operation that creates the QP and is supplied as an identifying parameter for other QP operations.

Queue Pair Number Identifies a specific Queue Pair within a Channel Adapter.

R_Key See Remote Key.

Raw Datagram A packet that contains an IBA Local Route Header, may contain an IBA Global Route Header, but does not contain an IBA Transport header, and is not handled by IBA transport services.

RC See Reliable Connection.

RD See Reliable Datagram.

RDC See Reliable Datagram Channel.

RDD See Reliable Datagram Domain.

RDETH Reliable Datagram Extended Transport Header.

RDMA See Remote Direct Memory Access.

Receive Queue One of the two queues associated with a Queue Pair. The receive queue contains Work Queue Elements that describe where to place incoming data.

Region Handle See Memory Region Handle.

Registered Memory A region of memory that has been through Memory Registration.

Registration See Memory Registration.

Registered memory region See Memory Registration.

Reliable Connection A Transport Service Type in which a Queue Pair is associated with only one other QP, such that messages transmitted by the send queue of one QP are reliably delivered to receive queue of the other QP. As such, each QP is said to be “connected” to the opposite QP.

Reliable Datagram A Transport Service Type in which a Queue Pair may communicate with multiple other QPs over a Reliable Datagram Channel. A message transmitted by an RD QP’s send queue will be reliably delivered to the receive queue of the QP specified in the associated Work Request. Despite the name, Reliable Datagram messages are not limited to a single packet.

Reliable Datagram Channel — The association of two Reliable Datagram End to End Contexts. A Reliable Datagram channel may multiplex Reliable Datagrams from many RD Queue Pairs.

Reliable Datagram Domain — An association that defines which Reliable Datagram Queue Pairs may use an End to End Context.

Remote Direct Memory Access — Method of accessing memory on a remote system without interrupting the processing of the CPU(s) on that system.

Remote Key — An opaque object, created by a verb, referring to a Memory Registration or Memory Window, used with a Virtual Address to describe authorization for the remote device to access local memory. It may also be used by the HCA hardware to identify the appropriate page tables for use in translating virtual to physical addresses.

Retired — The state of a Work Queue Element after the Host Channel Adapter completes the operation specified by the WQE, but before the Work Completion has been presented to the consumer.

RNR Nak — Receiver Not Ready. A response signifying that the receiver is not currently able to accept the request, but may be able to do so in the future.

Router — A device that transports packets between IBA subnets.

SA — See Subnet Administration.

SAR — Segmentation and Re-assembly.

Send Queue — One of the two queues of a Queue Pair. The Send queue contains WQEs that describe the data to be transmitted.

Server

1) The passive entity in a connection establishment exchange.
2) An entity (e.g., a process) that provides services in response to requests from clients.

Service ID — A value that allows a Communication Manager to associate an incoming connection request with the entity providing the service. The Service ID is similar to the TCP Port Number.

Service Level — Value in the Local Route Header identifying the appropriate Virtual Lane for a packet, enabling the implementation of differentiated services. While the appropriate VL for a specific Service Level may differ over a packet's Path, the Service Level remains constant.

Service Type — See Transport Service Type.

Signaled Completion — A modifier used for Work Requests submitted to the Send Queue specifying that a Work Completion shall be generated when the work requested completes, whether successfully or in error.

SGID — Source Global Identifier.

SLID — Source Local Identifier

SL — See Service Level.

SM — See Subnet Manager.

SMA — See Subnet Management Agent.

SMP — See Subnet Management Packet.

Solicited Event — A facility by which a message sender may cause an event to be generated at the recipient when the message is received.

Subnet — A set of Infiniband™ Architecture Ports, and associated links, that have a common Subnet ID and are managed by a common Subnet Manager. Subnets may be connected to each other through routers.

Subnet Administration — The architectural construct that implements the interface for querying and manipulating subnet management data.

Subnet Manager — One of several entities involved in the configuration and control of the subnet.

> **Master Subnet Manager:** The subnet manager that is authoritative, that has the reference configuration information for the subnet.
>
> **Standby Subnet Manager:** A subnet manager that is currently quiescent, and not in the role of a master SM, by agency of the master SM. Standby SMs are dormant managers.

Subnet Management Agent — An entity present in all IBA Channel Adapters and Switches that processes Subnet Management Packets from Subnet Manager(s).

Subnet Management Data — Vital Product Data required by the Subnet Manager.

Subnet Management Packet — The subclass of Management Datagrams used to manage the subnet. SMPs travel exclusively over Virtual Lane 15 and are addressed exclusively to Queue Pair Number 0.

Switch — A device that routes packets from one link to another of the same Subnet, using the Destination Local Identifier field in the Local Route Header.

TCA — See Target Channel Adapter.

Target Channel Adapter
A Channel Adapter typically used to support I/O devices. TCAs are not required to support the Verbs interface. See also I/O Unit.

Transport Service Type
Describes the reliability, sequencing, message size, and operation types that will be used between the communicating Channel Adapters.

Transport service types that use the IBA transport:

- Reliable Connection
- Unreliable Connection
- Reliable Datagram
- Unreliable Datagram

Raw Datagram service does not use the IBA transport.

UC
See Unreliable Connection.

UD
See Unreliable Datagram.

Unicast
An identifier for a single port. A packet sent to a unicast address is delivered to the port identified by that address.

Unit
One or more sets of processes and/or functions attached to the fabric by one or more channel adapters. See Host and I/O Unit.

Unreliable Connection
A Transport Service Type in which a Queue Pair is associated with only one other QP, such that messages transmitted by the send queue of one QP are, if delivered, delivered to the receive queue of the other QP. As such, each QP is said to be “connected” to the opposite QP. Messages with errors are not retried by the transport, and error handling must be provided by a higher level protocol.

Unreliable Datagram
A Transport Service Type in which a Queue Pair may transmit and receive single-packet messages to/from any other QP. Ordering and delivery are not guaranteed, and delivered packets may be dropped by the receiver.

Unsignaled Completion
A modifier used for Work Requests submitted to the Send Queue signifying that a Work Completion is to be generated only if the requested action completes in error.

Variant CRC
A CRC covering all the fields of a packet, including those that may be changed by Switches.

VCRC
See Variant CRC.

Verbs
An abstract description of the functionality of a Host Channel Adapter. An

operating system may expose some or all of the verb functionality through its programming interface.

Verbs Consumer The direct user of the Verbs.

Virtual Lane A method of providing independent data streams on the same physical link.

Vital Product Data Device-specific data to support management functions.

VL See Virtual Lane.

VPD See Vital Product Data.

WC See Work Completion.

Window Handle An opaque object that identifies a Memory Window.

Work Completion The consumer-visible representation of a Completion Queue Entry. A Work Completion may be obtained when a consumer polls a Completion Queue.

Work Queue One of Send Queue or Receive Queue.

Work Queue Element The Host Channel Adapter's internal representation of a Work Request. The consumer does not have direct access to Work Queue Elements.

Work Queue Pair See Queue Pair.

Work Request The means by which a consumer requests the creation of a Work Queue Element.

WQ See Work Queue.

WQE Work Queue Element, commonly pronounced "wookie".

WQP See Work Queue Pair.

WR See Work Request.

CHAPTER 3: ARCHITECTURAL OVERVIEW

This chapter provides a top-down description of the InfiniBand™ Architecture (IBA) features, capabilities, components, and elements and it describes various principles of operation. It is a high level overview intended as an informative guide and thus certain details are intentionally excluded for the purpose of clarity.

IBA defines a System Area Network (SAN) for connecting multiple independent processor platforms (i.e., host processor nodes), I/O platforms, and I/O devices (see Figure 6). The IBA SAN is a communications and management infrastructure supporting both I/O and interprocessor communications (IPC) for one or more computer systems. An IBA system can range from a small server with one processor and a few I/O devices to a massively parallel supercomputer installation with hundreds of processors and thousands of I/O devices. Furthermore, the internet protocol (IP) friendly nature of IBA allows bridging to an internet, intranet, or connection to remote computer systems.

IBA defines a switched communications fabric allowing many devices to concurrently communicate with high bandwidth and low latency in a protected, remotely managed environment. An endnode can communicate over multiple IBA ports and can utilize multiple paths through the IBA fabric. The multiplicity of IBA ports and paths through the network are exploited for both fault tolerance and increased data transfer bandwidth.

IBA hardware off-loads from the CPU much of the I/O communications operation. This allows multiple concurrent communications without the traditional overhead associated with communicating protocols. The IBA SAN provides its I/O and IPC clients zero processor-copy data transfers, with no kernel involvement, and uses hardware to provide highly reliable, fault tolerant communications.

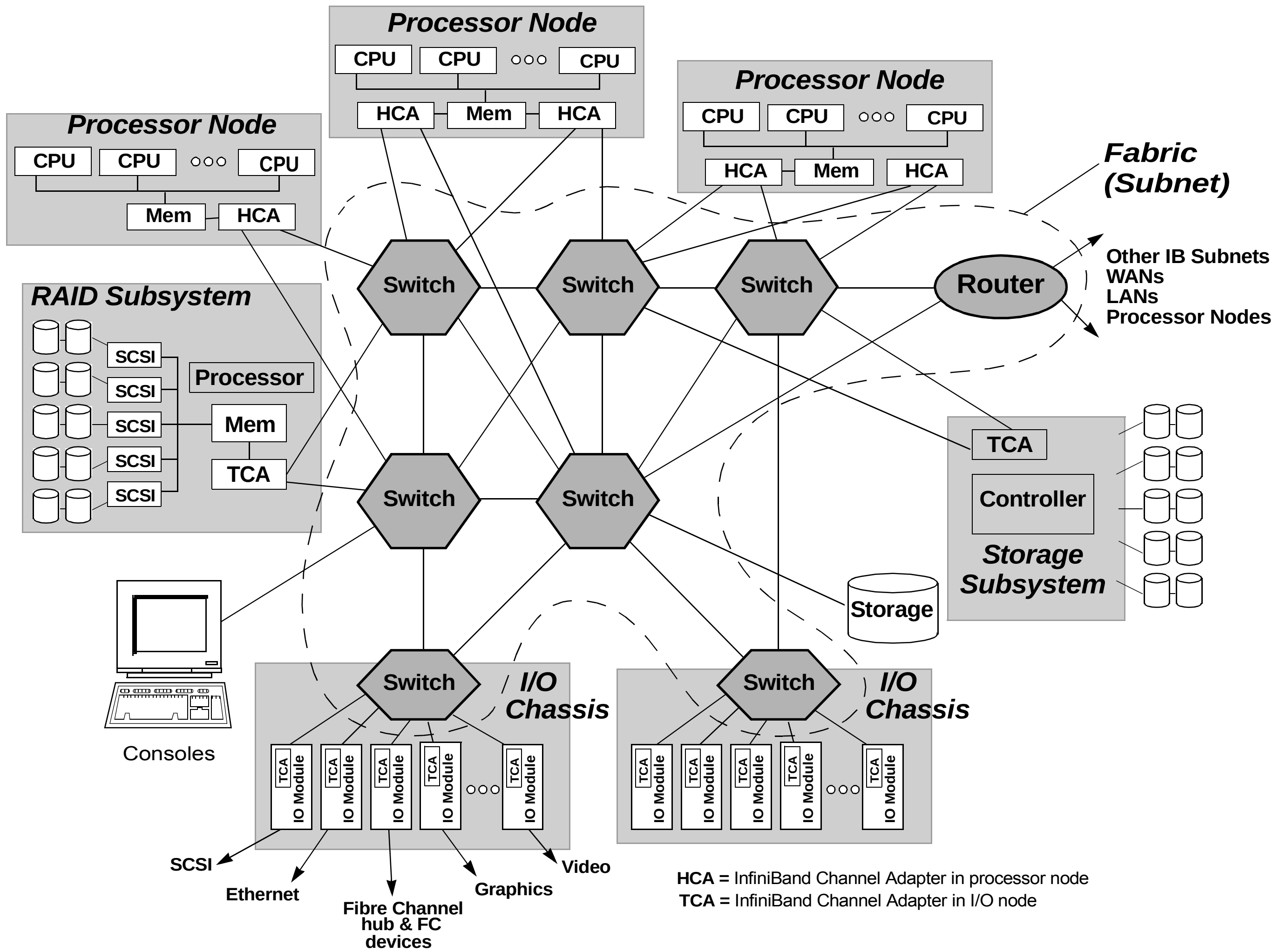


Figure 6 IBA System Area Network

An IBA System Area Network consists of processor nodes and I/O units connected through an IBA fabric made up of cascaded switches and routers.

IO units can range in complexity from single ASIC IBA attached devices such as a SCSI or LAN adapter to large memory rich RAID subsystems that rival a processor node in complexity.

3.1 ARCHITECTURE SCOPE

This volume of the InfiniBand Architecture Specification defines the interconnect fabric, routing elements, endnodes, management infrastructure, and the communication formats and protocols. It does not specify I/O commands or cluster services.

For example, consider an IBA SCSI adapter. IBA does not define the disk I/O commands, how the SCSI adapter communicates with the disk, how the operating system (OS) views the disk device, nor which node in the cluster owns the disk adapter. IBA is an essential underpinning of each of these operations, but does not directly define any of them. Instead, IBA defines how data and commands can be transported between the I/O driver on a processor node and the SCSI adapter.

IBA handles the data communications for I/O and IPC in a multi-computer environment. It supports the high bandwidth and scalability required for IO. It caters to the extremely low latency and low CPU overhead required for IPC. With IBA, the OS can provide its clients with communication mechanisms that bypass the OS kernel and directly access IBA network communication hardware, enabling efficient message passing operation. IBA is well suited to the latest computing models and will be a building block for new forms of I/O and cluster communication. IBA allows I/O units to communicate among themselves and with any or all of the processor nodes in a system. Thus an I/O unit has the same communications capability as any processor node.

3.1.1 TOPOLOGIES & COMPONENTS

At a high level, IBA serves as an interconnect for endnodes as illustrated in Figure 7. Each node can be a processor node, an I/O unit, and/or a router to another network.

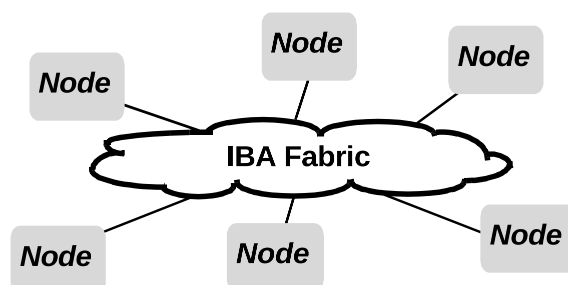


Figure 7 IBA Network

An IBA network is subdivided into subnets interconnected by routers as illustrated in Figure 8. Endnodes may attach to a single subnet or multiple subnets.

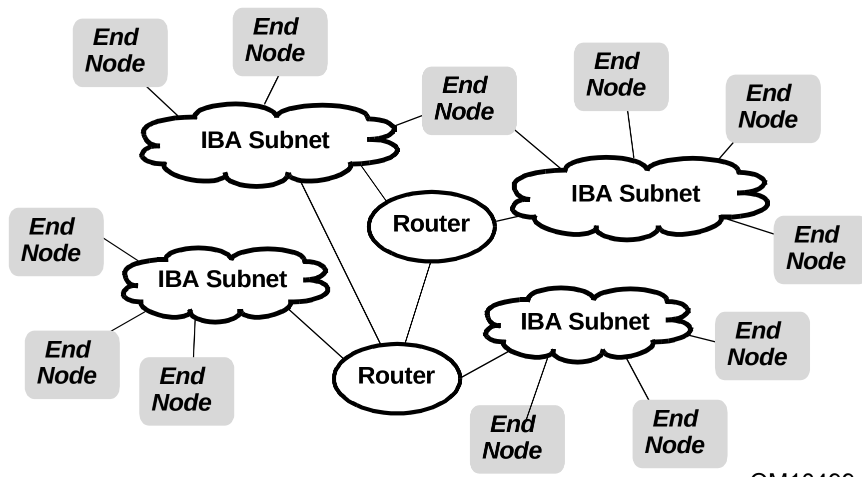


Figure 8 IBA Network Components

An IBA subnet is composed of endnodes, switches, routers, and subnet managers interconnected by links as illustrated in Figure 9. Each IBT device may attach to a single switch or multiple switches and/or directly with each other[1]. Multiple links can exist between any two IBT devices.

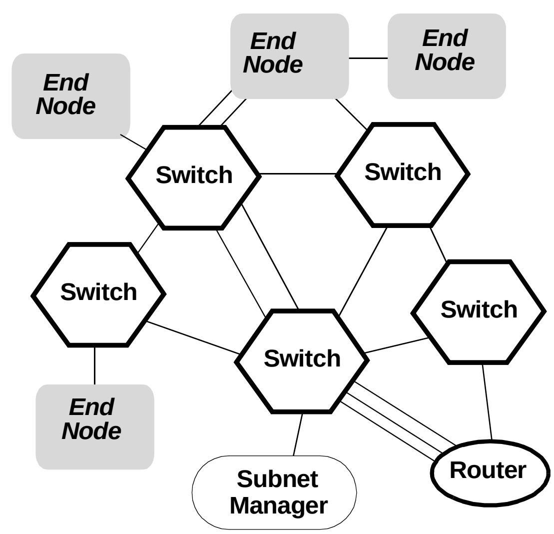


Figure 9 IBA Subnet Components

1. Single endnode to endnode connection creates an independent subnet, with no connectivity to the remainder of the IBT devices, in which case one of the two interconnected endnodes functions as the subnet manager for that link.

The architecture is optimized for units that contain multiple independent processes and threads (consumers) as illustrated in Figure 10. Each channel adapter constitutes a node on the fabric. The architecture supports multiple channel adapters per unit with each channel adapter providing one or more ports that connect to the fabric, in which case the processor node appears as multiple endnodes to the fabric.

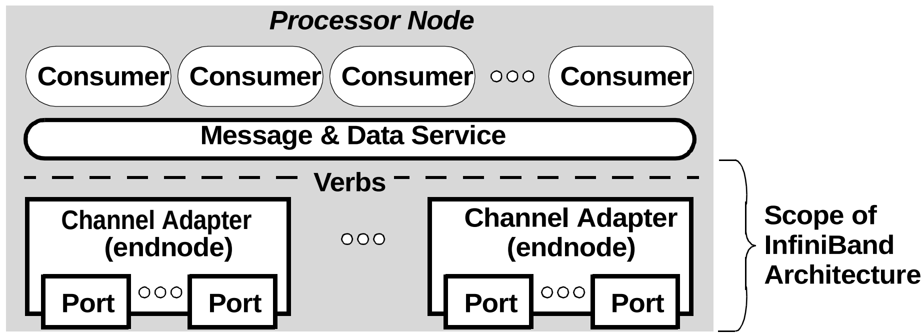


Figure 10 Processor Node

In a processor node, the message and data service is an OS component that is outside the scope of this document. This document specifies the semantic interface between the message and data service and a channel adapter. This semantic interface is referred to as IBA Verbs. Verbs describe the functions necessary to configure, manage, and operate a host channel adapter. These verbs identify the appropriate parameters that need to be included for each particular function. Verbs are not an API, but provide the framework for the OSV to specify the API.

IBA is architrected as a first order network and as such it defines the host behavior (verbs) and defines memory operation such that the channel adapter can be located as close to the memory complex as possible. It provides independent direct access between consenting consumers regardless of whether those consumers are I/O drivers and I/O controllers or software processes communicating on a peer to peer basis. IBA provides both channel semantics (send and receive) and direct memory access with a level of protection that prevents access by non participating consumers.

3.2 COMMUNICATION

3.2.1 QUEUING

The foundation of IBA operation is the ability of a consumer to queue up a set of instructions that the hardware executes. This facility is referred to as a work queue. Work queues are always created in pairs, called a Queue Pair (QP), one for send operations and one for receive operations. In general, the send work queue holds instructions that cause data to be transferred between the consumer's memory and another consumer's memory, and the receive work queue holds instructions about where to place data that is received from another consumer. The other consumer is referred to as a *remote consumer* even though it might be located on the same node. IBA specifically describes the queuing relationship for a *Host Channel Adapter* (HCA) but not the I/O unit because an I/O unit is not necessarily subject to 2nd and 3rd party interoperability that is present in a host environment (i.e., interoperability between the HCA vendor, the OS vendor, and an IHV's I/O driver or an ISV's application using IPC). The following describes the HCA queuing model.

The consumer submits a work request (WR), which causes an instruction called a Work Queue Element (WQE) to be placed on the appropriate work queue. The channel adapter executes WQEs in the order that they were placed on the work queue. When the channel adapter completes a WQE, a Completion Queue Element (CQE) is placed on a completion queue[1]. Each CQE specifies all the information necessary for a work completion, and either contains that information directly or points to other structures, for example, the associated WQE, that contain the information.

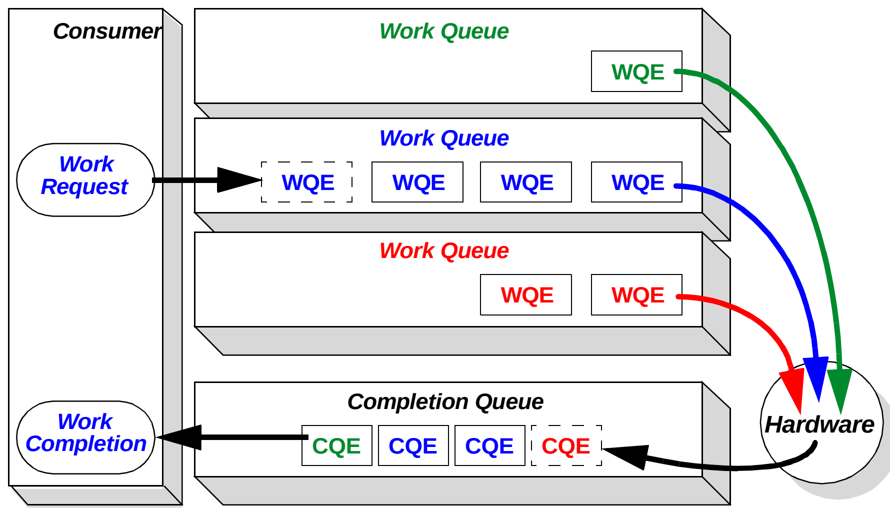


Figure 11 Consumer Queuing Model

Each consumer may have its own set of work queues, each pair of work queues is independent from the others. Each consumer creates one or more completion queues and associates each send and receive queue to a particular completion queue. It is not necessary that both the send and receive queue of a work queue pair use the same completion queue.

Because some work queues require an acknowledgment from the remote node and some WQEs use multiple packets to transfer the data, the channel adapter can have multiple WQEs in progress at the same time, even from the same work queue. Thus the order in which CQEs are

1. WQEs and CQEs are not architected entities, only the Work Request verbs are architected.

posted to the completion queue is not deterministic except that CQEs for the same work queue are normally posted in the order that the corresponding WQE was posted to the work queue[1].

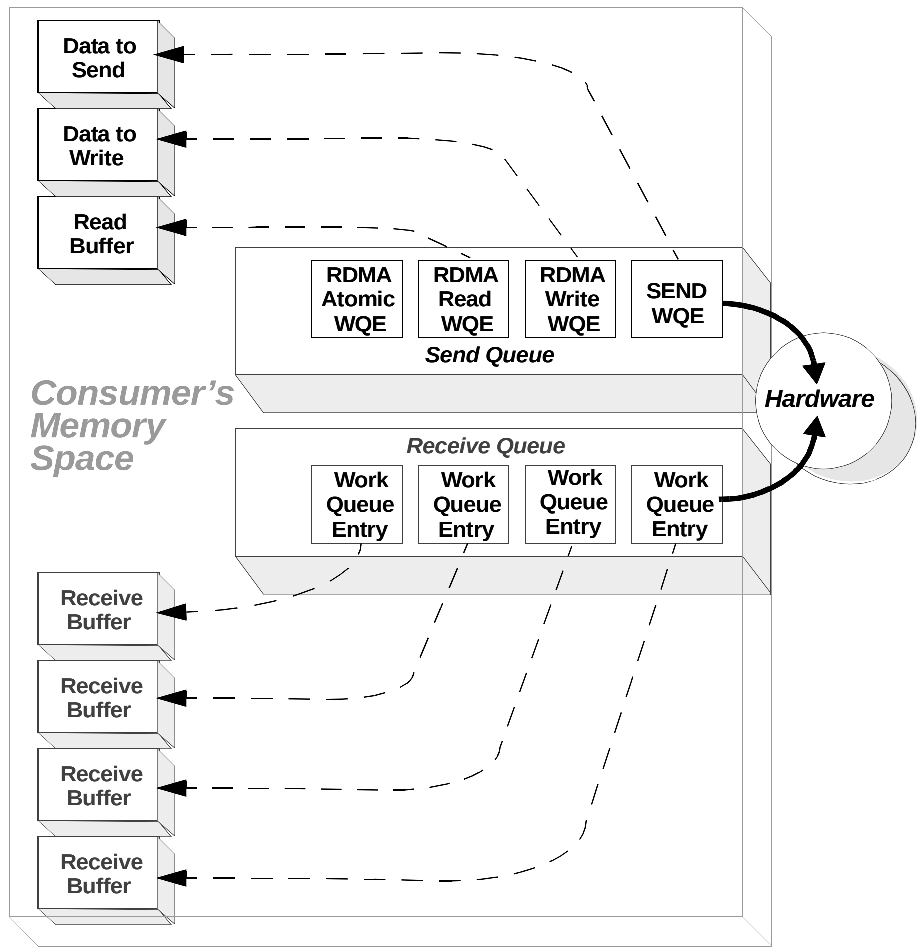


Figure 12 Work Queue Operations

There are three classes of send queue operations SEND, Remote memory Access (RDMA), and MEMORY BINDING.

- For a SEND operation, the WQE specifies a block of data in the consumer's memory space for the hardware to send to the destination, letting a receive WQE already queued at the destination specify where to place that data.

1. Receive completions for reliable datagram service are the exception because concurrent reception on multiple EE contexts can result in out of order posting.

- For an RDMA operation, the WQE also specifies the address in the remote consumer's memory. Thus an RDMA operation does not need to involve the receive work queue of the destination[1]. There are 3 types of RDMA operations, RDMA-WRITE, RDMA-READ, and ATOMIC.
    - The RDMA-WRITE operation stipulates that the hardware is to transfer data from the consumer's memory to the remote consumer's memory.
    - The RDMA-READ operation stipulates that the hardware is to transfer data from the remote memory to the consumer's memory.
    - The ATOMIC operation stipulates that the hardware is to perform a read of a remote 64-bit memory location. The target returns the value read, and conditionally modifies/replaces the remote memory contents by writing an updated value back to the same location.
- MEMORY BINDING instructs the hardware to alter memory registration relationships (see section 10.6.6.2). It associates (binds) a Memory Window to a specified range within an existing Memory Region. Memory binding allows a consumer to specify which portions of registered memory it shares with other nodes (i.e., the memory a remote node can access) and specifies read and write permissions. The result produces a memory key (R_KEY) that the consumer passes to remote nodes for their use in their RDMA operations.

There is only one receive queue operation and it is to specify a receive data buffer.

- A RECEIVE WQE specifies where the hardware is to place data received from another consumer when that consumer executes a SEND operation. Each time the remote consumer successfully executes a SEND operation, the hardware takes the next entry from the receive queue, places the received data in the memory location specified in that receive WQE, and places a CQE on the completion queue indicating to the consumer that the receive operation has completed. Thus the execution of a SEND operation causes a receive queue operation at the remote consumer.

Normally an RDMA operation does not consume a receive WQE at the destination, but there is one exception. That is for an RDMA WRITE operation which specifies immediate data. *Immediate data* is 32 bits of information that is optionally provided in a SEND or RDMA WRITE instruction, transferred as part of the operation, but instead of writing the immediate data to memory, the data is treated as another piece of status information

1. RDMA Write with immediate data does involve the destination's receive work queue.

and returned as a special field of the RECEIVE CQE status. This means that an RDMA WRITE with immediate data will consume a RECEIVE WQE at the destination.

3.2.2 CONNECTIONS

IBA supports both connection oriented and datagram service. For connected service, each QP is associated with exactly one remote consumer. In this case the QP context is configured with the identity of the remote consumer's queue pair. The remote consumer is identified by a port and a QP number. The port is identified by a local ID (LID) and optionally a Global ID (GID). During the communication establishment process, this and other information is exchanged between the two nodes.

For datagram service, a QP is not tied to a single remote consumer, but rather information in the WQE identifies the destination. A communication setup process similar to the connection setup process needs to occur with each destination to exchange that information.

3.3 COMMUNICATIONS STACK

The communication stack for IBA is illustrated in Figure 13. The architecture provides a number of IBA transactions that a consumer can use to execute a transaction with a remote consumer. The consumer posts work queue elements (WQE) to the QP and the channel adapter interprets each WQE to perform the operation.

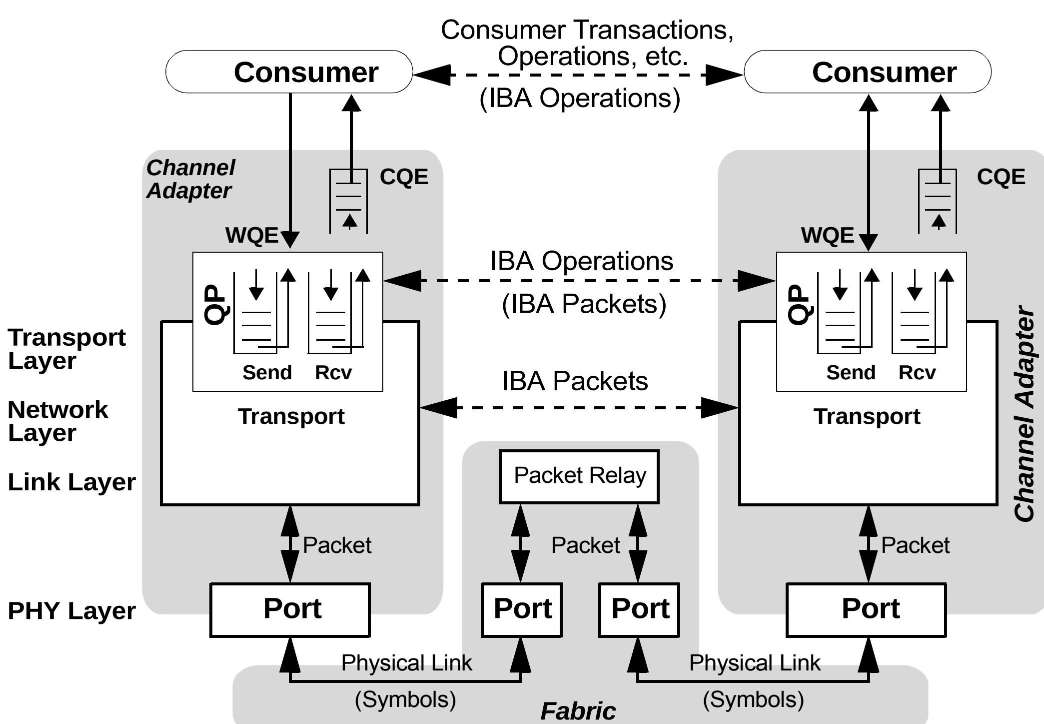


Figure 13 IBA Communication Stack

For Send Queue operations, the channel adapter interprets the WQE, creates a request message. segments the message into multiple packets if necessary, adds the appropriate routing headers, and sends the packet out the appropriate port.

The port logic transmits the packet over the link where switches and routers relay the packet through the fabric to the destination.

When the destination receives a packet, the port logic validates the integrity of the packet. The channel adapter associates the received packet with a particular QP and uses the context of that QP to process the packet and execute the operation. If necessary, the channel adapter creates a response (acknowledgment) message and sends that message back to the originator.

Reception of certain request messages cause the channel adapter to consume a WQE from the receive queue. When it does, a CQE corresponding to the consumed WQE is placed on the appropriate completion queue, which causes a work completion to be issued to the consumer that owns the QP.

3.4 IBA COMPONENTS

The devices in an IBA system are classified as:

- switches
- routers
- channel adapters
- repeaters
- links that interconnect switches, routers, repeaters, and channel adapters

The management infrastructure includes:

- subnet managers
- general service agents

3.4.1 LINKS & REPEATERS

Links interconnect channel adapters, switches, repeaters, and routing devices to form a fabric. A link can be a copper cable, an optical cable, or printed circuit wiring on a backplane. Repeaters are transparent[1] devices that extend the range of a link. Volume 2 of InfiniBand Architecture specifies link and repeater requirements for various media types as well as de-

1. Transparent in the sense repeaters only participate at the physical layer protocol level and nodes are not aware of their presence.

fines various module form factors for I/O devices. The architecture described in Volume 1 is independent of the type of link and the form factor.

Links and repeaters are not directly addressable but the link status can be determined via the device on each end of the link.

3.4.2 CHANNEL ADAPTERS

Channel adapters are the IBA devices in processor nodes and I/O units that generate and consume packets. IBA defines two types of channel adapters: *Host Channel Adapter* (HCA) and *Target Channel Adapter* (TCA). The HCA provides a consumer interface providing the functions specified by IBA verbs. IBA does not specify the semantics of the consumer interface for a TCA.

A channel adapter is a programmable DMA engine with special protection features that allow DMA operations to be initiated locally and remotely.

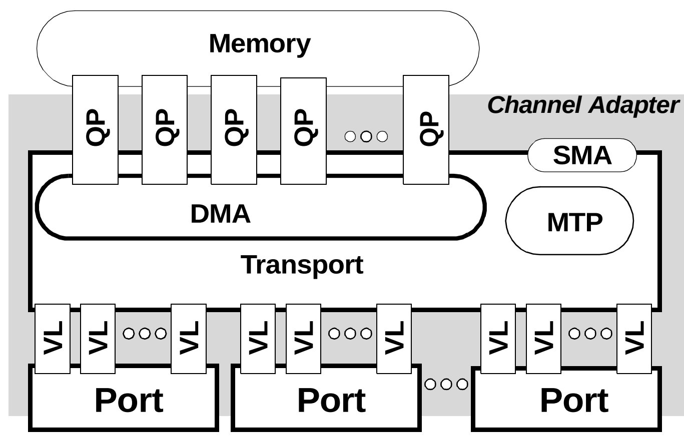


Figure 14 Channel Adapter

A channel adapter may have multiple ports. Each port of a channel adapter is assigned a *Local ID* (LID) or a range of LIDs. Each port has its own set of transmit and receive buffers such that each port is capable of sending and receiving concurrently. Buffering is channeled through *virtual lanes* (VL) where each VL has its own flow control.

The channel adapter provides a Memory Translation & Protection (MTP) mechanism that translates virtual addresses to physical addresses and to validate access rights. Specific memory management mechanisms are not specified by this document, and requirements for such mechanisms are not specified for TCAs.

The channel adapter provides multiple instances of the communication interface to its consumer in the form of *queue pairs* (QP) comprised of a send and receive work queue.

A subnet manager configures channel adapters with the local addresses for each physical port, i.e., the port's LID. The entity that communicates with the subnet manager for the purpose of configuring the channel adapter is referred to as the *Subnet Management Agent* (SMA).

Each channel adapter has a *globally unique identifier* (GUID) assigned by the channel adapter vender. Since local IDs assigned by the subnet manager are not persistent (i.e., might change from one power cycle to the next), the channel adapter GUID (called Node GUID) becomes the primary object to use for persistent identification of a channel adapter. Additionally, each port has a Port GUID assigned by the channel adapter vender.

3.4.3 SWITCHES

In contrast to channel adapters, switches do not generate nor consume packets (except for management packets). They simply pass them along based on the destination address in the packet's local route header.

IBA switches are the fundamental routing component for intra-subnet routing (inter-subnet routing is provided by IBA routers). Switches interconnect links by relaying packets between the links.

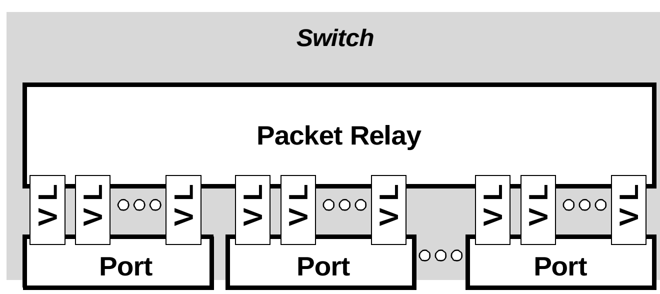


Figure 15 IBA Switch Elements

Switches expose two or more ports between which packets are relayed.

Switches are transparent to the endnodes which means they are not directly addressed (except for management operations). Instead, packets transverse the switch fabric virtually unchanged by the fabric. To this end, every destination within the subnet is configured with one or more unique local identifiers (LID). From the point of view of a switch, a LID represents a path through the switch. Switch elements are configured with forwarding tables. Packets contain a destination address that specifies the LID of the destination. Individual packets are forwarded within a switch to an outbound port or ports based on the packet's Destination LID and the Switch's forwarding table.

IBA switches support unicast forwarding and may support multicast forwarding. Unicast is the delivery of a single packet to a single destination and multicast is the ability of the fabric to deliver a single packet to multiple destinations.

A subnet manager configures switches including loading their forwarding tables.

To maximize availability, multiple paths between endnodes may be deployed within the switch fabric. If multiple paths are available between switches, the subnet manager can use these paths for redundancy or for destination LID based load sharing. Where multiple paths exists, a subnet manager can re-route packets around failed links by re-loading the forwarding tables of switches in the affected area of the fabric.

3.4.4 ROUTERS

Like switches, routers do not generate nor consume packets (except for management packets). They simply pass them along. Routers forward packets based on the packet's global route header and actually replaces the packet's local route header as the packet passes from subnet to subnet.

IBA routers are the fundamental routing component for inter-subnet routing (intra-subnet routing is provided by IBA switches). Routers interconnect subnets by relaying packets between the subnets.

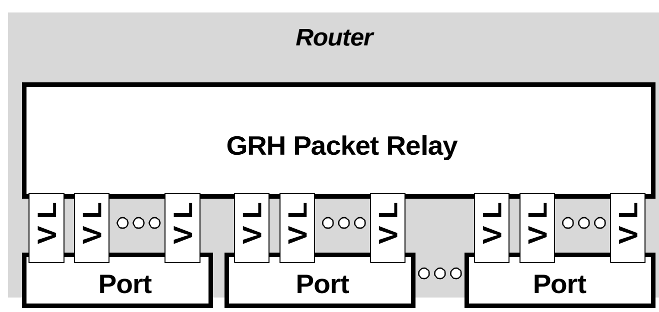


Figure 16 IBA Router Elements

Routers expose one or more ports between which packets are relayed. Routers could be embedded with other devices, such as channel adapters or switches.

Routers are not completely transparent to the endnodes since the source must specify the LID of the router and also provide the GID of the destination.

Each subnet is uniquely identified with a subnet ID known as the Subnet Prefix. The subnet manager programs all ports (via the PortInfo attribute) with the Subnet Prefix for that subnet. When combined with a Port GUID, this combination becomes a port's natural GID. Ports may have other locally administrated GIDs.

From the point of view of a router, the subnet prefix portion of a GID represents a path through the router. IPv6 specifies the protocol performed between routers to derive their forwarding tables. Individual packets are forwarded within a router to an outbound port or ports based on the packet's Destination GID and the router's forwarding table.

Each router forwards the packet through the next subnet to another router until the packet reaches the target subnet. The last router sends the packet using the LID associated with the Destination GID as the Destination LID.

A subnet manager configures routers with information about the subnet such as which VLs to use and partition information.

To maximize availability, multiple paths between subnets may be deployed within the fabric. If multiple paths are available, routers might use those paths for redundancy or for load sharing. Where multiple paths exist, a router can re-route packets around failed subnets.

3.4.5 Management Components

IBA management provides for a subnet manager and an infrastructure that supports a number of general management services. The management infrastructure requires a subnet management agent in each node and defines a general service interface that allows additional general services agents.

The architecture defines a common management datagram (MAD) message structure for communicating between managers and management agents.

3.4.5.1 Subnet Managers

A *Subnet Manager* (SM) is an entity attached to a subnet that is responsible for configuring and managing switches, routers, and channel adapters. A SM can be implemented with other devices, such as a channel adapter or a switch.

IBA supports the notion of multiple subnet managers per subnet and specifies how multiple subnet managers negotiate for one to become the master SM. It does not prohibit other methods between cooperating SMs for governing master/standby relationships

The master SM:

- discovers the subnet topology,
- configures each channel adapter port with a range of LIDs, GIDs subnet prefix, and P_Keys,
- configures each switch with a LID, the subnet prefix, and with its forwarding database,
- maintains the endnode and service databases for the subnet and thus provides a GUID to LID/GID resolution service as well as a services directory.

3.4.5.2 SUBNET MANAGEMENT AGENTS

Each node provides a Subnet Management Agent (SMA) that the SM access through a well known interface called the Subnet Management Interface (SMI). SMI allows for both LID Routed packets and Directed Routed packets. Directed routing provides the means to communicate before switches and end nodes are configured. Only the SMI allows for directed routed packets.

3.4.5.3 GENERAL SERVICE AGENTS

Each node may contain additional management agents referred to as General Service Agents (GSA*) that can be accessed through a well known interface called the General Service Interface (GSI). The GSI only supports LID routing. The general service classes defined by IBA are:

- Subnet Administration (SubnAdm) - this is a service provided by the SM that allows nodes to access information about the subnet to discover other nodes and services, to resolve paths, and to register its services.
- Performance Management (Perf) - monitors and reports well-defined performance counters
- Device Management (DevMgt) - provides for management of I/O devices behind TCAs.
- Baseboard Management (BM) - provides for chassis management using IB-ML as defined in Volume 2.
- SNMP Tunneling (SNMP) - provides SNMP functionality by defining the method for sending and receiving SNMP messages.
- Vendor Defined (Vendor) - allows private extensions that a device vendor may use to remotely configure and manage its devices.

- Communication Management (ConMgt) - Provides for connection establishment and other communication management functions between endnodes.
- Device Configuration (DevConfMgt) - Provides I/O resource management

3.5 IBA Features

3.5.1 Queue Pairs

The QP is the virtual interface that the hardware provides to an IBA consumer and it provides a virtual communication port for the consumer. The architecture supports up to 2^{24} QPs per channel adapter and the operation on each QP is independent from the others. Each QP provides a high degree of isolation and protection from other QP operations and other consumers. Thus a QP can be considered a private resource assigned to a single consumer. A consumer might consume multiple QPs as illustrated in Figure 17.

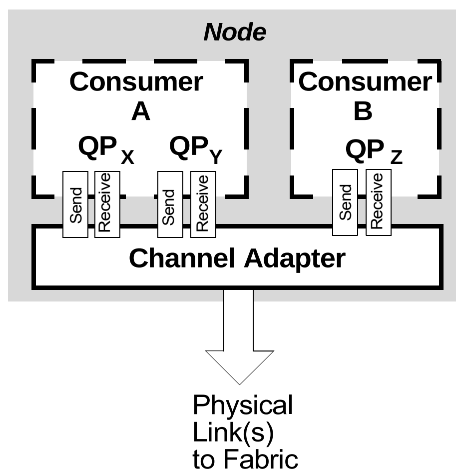


Figure 17 Communication Interface

The consumer creates this virtual communication port by allocating a QP and specifying its class of service. Communication takes place between a source QP and a destination QP. For connection oriented service, each QP is tightly bound to exactly one other QP, usually on a different node. The consumer initiates any communication establishment necessary to bind the QP with the destination QP and configures the QP context with certain information such as Destination LID, service level, and negotiated operating limits.

The consumer posts work requests to a QP to invoke communication through that QP.

3.5.2 TYPES OF SERVICE

Each QP is configured for a certain class of operation (referred to as service type) based on how the sourcing and receiving QPs interact. Both the source and destination QPs must be configured for the same service type. Each service type is based on the following attributes.

- **Connection oriented** versus **datagram -** For connection oriented service, the QP is associated with exactly one other QP and all work requests posted to the QP results in a message sent to the established destination QP. Datagram operation allows a single QP to be used to send and receive messages to/from any appropriate QP on any node.
- **Acknowledged** versus **unacknowledged -** For acknowledged service, a QP returns response messages when it receives request messages. Response messages might be positive acknowledgment (ACK), negative acknowledgment (NAK), or contain response data. Acknowledged service is referred to as *reliable* since the transport protocol guarantees un-corrupted data delivery, in order, exactly once. Unacknowledged service is referred to as unreliable because the transport protocol does not guarantee that all data is delivered. It does guarantee that all data is delivered at most once, and delivered data is not corrupted. Also there are certain cases where changes in fabric configuration might cause data to be delivered out of order.
- **IBA transport** versus **other transport -** IBA transport services define a specific transport protocol for channel based and memory based operations. IBA also supports using the channel adapter as a data link engine to send raw packets between nodes which is useful for supporting legacy protocol stacks and legacy networks.

The service types defined by IBA are specified in Table 2

Table 2 Service Types

Service Type	Connection Oriented	Acknowledged	Transport
Reliable Connection	yes	Yes	IBA
Unreliable Connection	yes	no	IBA
Reliable Datagram	no	Yes	IBA
Unreliable Datagram	no	no	IBA
RAW Datagram	no	no	Raw

Certain IBA operations are valid only over certain classes of service. A QP rejects a WQE for an operation that is not valid for the configured class of service.

Connection oriented service requires that the consumer initiate a communication establishment procedure (connection setup) with the target node to associate the QPs and establish QP context prior to any QP operation. Actually, all service classes except for raw datagram need some form of communication setup to associate queue pairs. For reliable datagram service, the node performs a communication establishment process to associate an end-to end (EE) context (explained later) with each target node. All QPs configured for Reliable Datagram service use established EE contexts and the work request specifies which EE context to use for that operation.

Raw Datagrams are similar to unreliable datagrams, except that the source QP does not know the identity of the QP that will receive and process the message. Raw datagrams allow for routers that forward raw datagram packets to non IBA destinations on a disparate fabric (such as a LAN or WAN) that has no equivalent of a QP. There are two types of raw datagrams, IPv6 and Ethertype. IPv6 raw datagrams contain a global routing header and the packet payload contains a transport protocol service data unit as identified in the global routing header. An Ethertype raw datagram contains an Ethernet Type field and the packet payload contains a transport protocol service data unit as identified in the Ethernet Type field.

IBA defines both channel (send/receive) and memory (RDMA) semantics. Raw datagram and Unreliable Datagram services do not support memory semantics.

3.5.3 Keys

IBA uses various keys to provide isolation and protection. Keys are values assigned by an administrative entity that are used in messages in various ways. The keys themselves do not provide security since the keys are available in messages that cross the fabric and thus any entity that can get to the interior of the fabric can ascertain key values. IBA does place restrictions on how applications can access certain keys.

The keys are:

- **Management Key** (M_Key): Enforces the control of a master subnet manager. Administered by the subnet manager and used in certain subnet management packets. Each channel adapter port has a M_Key that the SM sets and then enables. The SM may assign a different key to each port. Once enabled, the port rejects certain management packets that do not contain the programmed M_Key. Thus only a SM with the programed M_Key can alter a node's fabric configuration. The SM can prevent the port's M_Key from being read as long as the SM is active. The port maintains a time-out such that the port reverts to an unmanaged state if the SM fails. There is one M_Key for a switch.

- **Baseboard Management Key** (B_Key): Enforces the control of a subnet baseboard manager. Administered by the subnet baseboard manager and used in certain MADs. Each channel adapter port has a B_Key that the baseboard manager sets. The baseboard manager may assign a different key to each port. Once enabled, the port rejects certain management packets that do not contain the programmed B_Key. Thus only a baseboard manager with the programed B_Key can alter a node's baseboard configuration. The baseboard manager can prevent the port's B_Key from being read as long as the baseboard manager is active. The port maintains a time-out such that the port reverts to an unmanaged state if the baseboard manager fails. There is one B_Key for a switch.
- **Partition Key** (P_Key): Enforces membership. Administered through the subnet manager by the partition manager (PM). Each channel adapter port contains a table of partition keys which is setup by the PM. QPs are required to be configured for the same partition to communicate (except QP0, QP1, and ports configured for raw datagrams) and thus the P_Key is carried in every IB transport packet. Part of the communication establishment process determines which P_Key that a particular QP or EEC uses. An EEC contains the P_Key for Reliable Datagram service and a QP context contains the P_Key for the other IBA transport types. The P_Key in the QP or EEC is placed in each packet sent, and compared with the P_Key in each packet received. Received packets whose P_Key comparison fails are rejected. Each switch has one P_Key table for management messages and may optionally support partition enforcement tables that filter packets based on their P_Key.
- **Queue Key** (Q_Key): Enforces access rights for reliable and unreliable datagram service (RAW datagram service type not included). Administered by the channel adapter. During communication establishment for datagram service, nodes exchange Q_Keys for particular queue pairs and a node uses the value it was passed for a remote QP in all packets it sends to that remote QP. Likewise, the remote node uses the Q_Key it was provided. Receipt of a packet with a different Q_Key than the one the node provided to the remote queue pair means that packet is not valid and thus rejected.

  Q_Keys with the most significant bit set are considered controlled Q_Keys (such as the GSI Q_Key) and a HCA does not allow a consumer to arbitrarily specify a controlled Q_Key. An attempt to send a controlled Q_Key results in using the Q_Key in the QP context. Thus the OS maintains control since it can configure the QP context for the controlled Q_Key for privileged consumers.

- **Memory Keys** (L_Key and R_Key): Enables the use of virtual addresses and provides the consumer with a mechanism to control access to its memory. These keys are administered by the channel adapter through a registration process. The consumer registers a region of memory with the channel adapter and receives an L_Key and R_Key. The consumer uses the L_Key in work requests to describe local memory to the QP and passes the R_Key to a remote consumer for use in RDMA operations. When a consumer queues up a RDMA operation it specifies the R_Key passed to it from the remote consumer and the R_Key is included in the RDMA request packet to the original channel adapter. The R_Key validates the sender's right to access the destination's memory and provides the destination channel adapter with the means to translate the virtual address to a physical address.

3.5.4 VIRTUAL MEMORY ADDRESSES

IBA is optimized for virtual addressing. That is, an IBA consumer uses virtual addresses in work requests and the channel adapter is able to convert the virtual address to physical address as necessary. For this to happen, each consumer registers regions of virtual memory with the channel adapter and the channel adapter returns 2 memory handles called L_Key and R_Key to the consumer. The consumer then uses the L_key in each work request that requires a memory access to that region. See 3.5.3 for description of L_Key usage.

Memory Registration provides mechanisms that allow IBA consumers to de-scribe a set of virtually contiguous memory locations or a set of physically contiguous memory locations to allow the HCA to access the memory as a virtually contiguous buffer using virtual addresses.

IBA also supports remote memory access (RDMA) that permits a remote consumer to access that registered memory. For RDMA, the consumer passes the R_KEY and a virtual address of a buffer in that memory region to another consumer. That remote consumer supplies that R_Key in its RDMA WQEs that will access memory in the original node. See 3.5.3 for detailed description of R_Key usage.

3.5.5 PROTECTION DOMAINS

Not only does memory registration allow the use of virtual memory addressing, but it also provides an increased level of protection against inadvertent and unauthorized access.

Since a consumer might communicate with many different destinations but not wish to let all those destinations have the same access to its registered memory, IBA provides protection domains. Protection domains allow a consumer to control which set of its Memory Regions and Memory Windows can be accessed by which set of its QPs.

Before a consumer allocates a QP or registers memory, it creates one or more protection domains. QPs are allocated to, and memory registered to, a protection domain. L_Keys and R_Keys for a particular memory domain are only valid on QPs created for the same protection domain.

3.5.6 PARTITIONS

Partitioning enforces isolation among systems sharing an InfiniBand fabric. Partitioning is not related to boundaries established by subnets, switches, or routers. Rather a partition describes a set of endnodes within the fabric that may communicate.

Each port of an endnode is a member of at least one partition and may be a member of multiple partitions. A partition manager assigns partition keys (P_Keys) to each channel adapter port. Each P_Key represents a partition. Each QP[1] and EE context is assigned to a partition and uses that P_Key in all packets it sends and inspects the P_Key in all packets it receives. Reception of an Invalid P_Key causes the packet to be discarded.

Switches and routers may optionally be used to enforce partitioning. In this case the partition manager programs the switch or router with P_Key information and when the switch or router detects a packet with an invalid P_Key, it discards the packet.

3.5.7 VIRTUAL LANES

Virtual lanes (VL) provide a mechanism for creating multiple virtual links within a single physical link. A virtual lane represents a set of transmit and receive buffers in a port. All ports support VL_{15} which is reserved exclusively for subnet management. There are 15 other VLs (VL_0 to VL_{14}) called data VLs and all ports support at least one data VL (VL_0) and may provide VL_1 to VL_{n-1}, where n is the number of data VLs the port supports).

The actual data VLs that a port uses is configured by the SM and is based on the Service Level (SL) field in the packet. The default is to use VL_0 until the SM determines the number of VLs that are supported by both ends of the link and programs the port's SL to VL mapping table.

1. Except QP0, QP1, and QPs configured for Raw Datagrams type of service.

The port maintains separate flow control over each data VL such that excessive traffic on one VL does not block traffic on another VL.

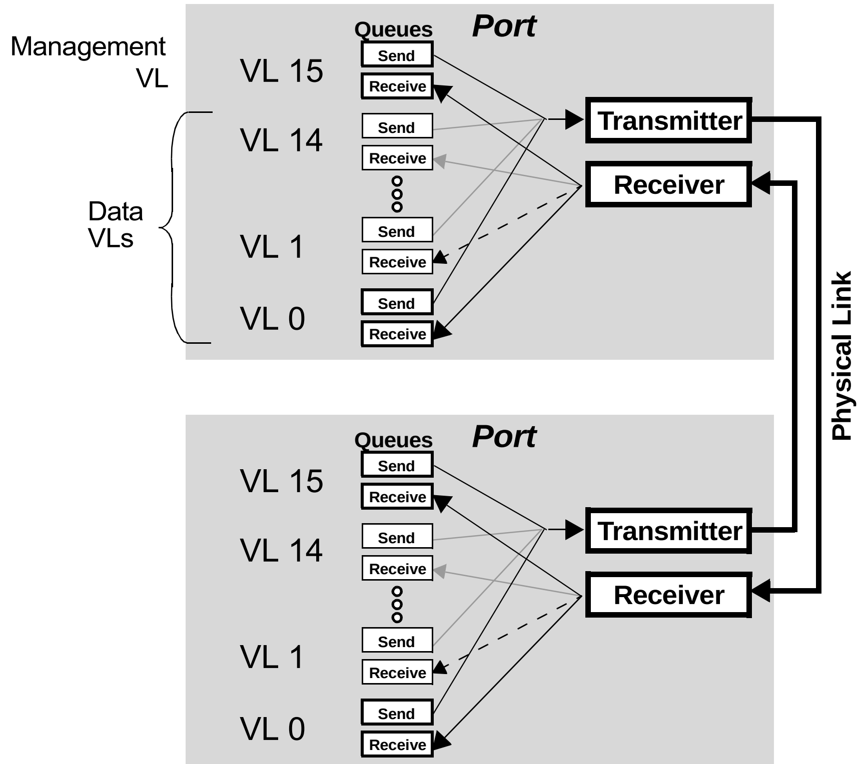


Figure 18 Virtual Lanes

VL assignment exists only between ports at each end of a link and VL assignment on one link is independent of assignments on other links.

Each packet has a SL which is specified in the packet header. As a packet traverses the fabric, its SL determines which VL will be used on each link. Each port maintains a table of SL to VL mapping such that a packet is sent on the appropriate VL.

When the ports at each end of a link support a different number of data VLs, the port with the higher number degrades to the number supported by the other port. Thus for ports that only support a single data VL, all data traffic defaults to VL_0.

3.5.8 QUALITY OF SERVICE

IBA provides several mechanisms that permit a subnet manager to administer various quality of service guarantees for both connected and connectionless services. These mechanisms are Service Level, Service Level to Virtual Lane Mapping, and Partitions. IBA does not define quality of service (QoS) levels (e.g., best effort).

3.5.8.1 SERVICE LEVEL

IBA defines a Service Level (SL) attribute that permits a packet to operate at one of 16 service levels. The definition and purpose of each service level is outside the scope of the architecture and left as a fabric administration policy. Thus the assignment of service levels is a function of each node's communication manager and its negotiation with a subnet manager.

3.5.8.2 SL TO VL MAPPING

Another IBA mechanisms that is tied to service levels is virtual lanes. Each packet identifies its SL and as the packet traverses the fabric, the packet's SL determines which VL is used on the next link. To this end, each port (switches, routers, endnodes) has a SL to VL mapping table that is configured by subnet management. Naturally, for all links that terminate at a port that only supports one data VL, all SLs map to VL_0. Otherwise, subnet management policy determines the mapping of each SL to an available VL.

Packets addressed to QP0 are Subnet Management Packets (SMP) and exclusively use VL15 and their SL is ignored. VL15 (the management VL) is not a data VL and is not used for packets not addressed to QP0.

3.5.8.3 PARTITIONS

Another IBA mechanism that can be tied to service levels is partitioning. Fabric administration can assign certain SLs for particular partitions. This allows the SM to isolate traffic flows between those partitions and even if both partitions operate at the same QoS level, each partition can be guaranteed its fair share of bandwidth regardless of whether nodes in other partitions misbehave or are over subscribed.

3.5.9 INJECTION RATE CONTROL

IBA defines a number of different link bit rates. The lowest bit rate of 2.5 Gb/sec is referred to as a 1x (times one) link. Other link rates are 10Gb/sec (4x) and 30 Gb/sec (1x2). To support multiple link speeds within a fabric, IBA defines a *Static Rate Control* mechanism that prevents a port with a high speed link from overrunning the capacity of a port with a lower speed link.

As part of the path resolution process, the SubnAdm:PathRecord provides the node with MTU and rate information for the path. Path information is used since either a switch port or the endnode could be the limiting factor.

The example in Figure 19 illustrates that port A with a 12x link speed has the potential for injecting traffic at 3 times the capacity of port B and 12 times the capacity of ports C, D, or E. Additionally port B has the potential for injecting traffic at 4 times the capacity of port C, D, or E. Since traffic tends to be bursty, every time port A sends to one of the other ports, the fabric has a high probability of congesting. Link flow control keeps the fabric from loosing packets due to that congestion, but the back pressure will effect other paths that otherwise would not be congested.

IBA solves this problem by defining a static rate control mechanism for ports that operate at link speeds greater than 1x.

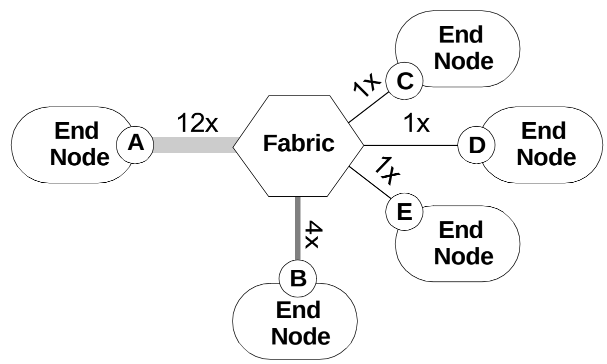


Figure 19 Rate Matching Example

Each destination has a time-out value associated with it and that time-out value is based on the ratio between the source and destination bit rates. When the source and destination bit rates are equal, the time-out values is 0 (not needed). Otherwise when the port transmits a packet to a destination, it puts that destination LID and a time-out value in its static rate control table. The port removes the entry after the time-out period expires. While the entry remains in the table, the port does not send any more packets to that destination (i.e., defers to traffic for other destinations not in the table). When there is no entry in the table, the port transmits the packet by placing it on the appropriate VL output queue.

3.5.10 ADDRESSING

Each endnode contains one or more channel adapters and each channel adapter contains one or more ports. Additionally each channel adapter contains a number of queue pairs (QP).

Each QP has a queue pair number (QPN) assigned by the channel adapter which uniquely identifies the QP within the channel adapter. There are two well-known QPs for each port (QP0 and QP1) and all other QPs are configured for operation through a particular port. For reliable datagram service, it is the EE context rather than the QP context that determines the port.

Packets other than raw datagrams contain the QPN of the destination QP. When the channel adapter receives a packet, it uses the context of the destination QPN (and EE context for reliable datagram) to process the packet.

Each port has a Local ID (LID) assigned by the local subnet manager (i.e., the subnet manager for the subnet). Within the subnet, LIDs are unique. Switches use the LID to route packets within the subnet. The local subnet manager configures routing tables in switches based on LIDs and where that port is located with respect to the specific switch. Each packet contains a Source LID (SLID) that identifies the port that injected the packet into the subnet and a Destination LID (DLID) that identifies the port where the fabric is to deliver the packet.

IBA also provides for multiple virtual ports within a physical port by defining a LID Mask Control (LMC). The LMC specifies the number of least significant bits of the LID that a physical port masks (ignores) when validating that a packet DLID matches its assigned LID. Those bits are not ignored by switches, therefore the subnet manager can program different paths through the fabric based on those least significant bits. Thus the port appears to be $2^{\mathbf{LMC}}$ ports for the purpose of routing across the fabric.

Each port also has at least one Global ID (GID) that is in the format of an IPv6 address. GIDs are globally unique. Each packet optionally contains a Global Route Header (GRH) specifying a Source GID (SGID) that identifies the port that injected the packet into the fabric and a Destination GID (DGID) that identifies the port where the fabric is to deliver the packet. Routers use the GRH to route packets between subnets. Switches ignore the GRH.

Each channel adapter has a Globally Unique Identifier (GUID) called the Node GUID assigned by the channel adapter vendor. Each of its ports has a Port GUID also assigned by the channel adapter vendor. The Port GUID combined with the local subnet prefix becomes a port’s default GID.

Subnet administration provides a GUID to LID/GID resolution service. Thus a node can persistently identify another node by remembering a Node or Port GUID.

The address of a QP is the combination of the port address (GID + LID) and the QPN. To communicate with a QP requires a vector of information including the port address (LID and/or GID), QPN, service level, path MTU, and possibly other information. This information can be obtained by a path query request addressed to Subnet Administration.

Service IDs are used to resolve QPs. Some Service IDs are well known (i.e., certain functions have a predetermined Service ID) and some are advertised in an I/O controller’s Service Entries list. The subnet manager provides the GUID to GID/LID resolution, but the target provides a Service ID to QP resolution as part of the communication management process.

In general, the target node of a Request for Communication message uses the Service ID to direct the request to the entity who decides whether to proceed with communication establishment. If the decision is affirmative, the target returns the information necessary to establish communication, which includes the QPN plus other information specific to the transport service type.

A simplified address resolution process is illustrated in Figure 20.

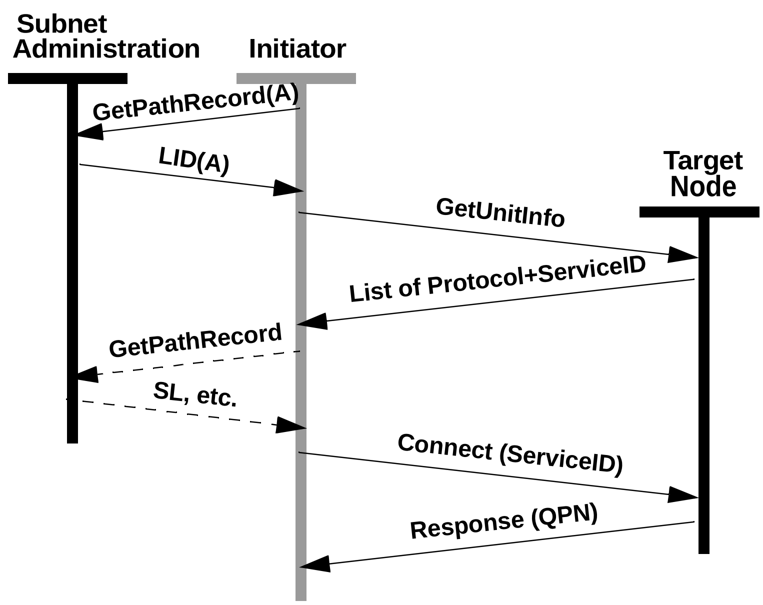


Figure 20 Simplified Address Resolution Process

In the illustration, the target is an I/O controller where the initiator learns the Service ID by querying the IOC for a list of I/O protocols supported. The second path resolution is only necessary if the service being established uses different path characteristics (SL, QoS, MTU, etc.) than the management MADs.

3.5.11 MULTICAST

Multicast is a one-to-many / many-to-many communication paradigm designed to simplify and improve the efficiency of communication between a set of endnodes.

Each multicast group is identified by a unique LID and GID. The LID is only unique within the subnet. A node joins and leaves a multicast group through a management action where the node supplies the LID for each port that will participate. This information is distributed to the switches. Each switch is configured with routing information for the multicast traffic which specifies all of the ports where the packet needs to travel. Care is taken to assure there are no loops (i.e., a single spanning tree such that a packet is not forwarded to a switch that already processed that packet).

The node uses the multicast LID and GID in all packets it sends to that multicast group. When a switch receives a multicast packet (i.e., a packet with a multicast LID in the packet's DLID field) it replicates the packet and sends it out to each of the designated ports except the arrival port. In this fashion, each cascaded switch replicates the packet such that the packet arrives only once at every subscribed endnode.

The channel adapter may limit the number of QPs that can register for the same multicast address. The channel adapter distributes multicast packets to QPs registered for that multicast address. A single QP can be registered for multiple addresses for the same port but if a consumer wishes to receive multicast traffic on multiple ports it needs a different QP for each port. The channel adapter recognizes a multicast packet by the packet's DLID or by the special value in the packet's Destination QP field and routes the packet to the QPs registered for that address and port. Note that the Destination QP field in a multicast packet is not a QPN.

3.5.11.1 MULTICAST EXAMPLE

Figure 21 illustrates an example unreliable multicast IBA operation:

- A packet with PSN = 1129 is received on an IBA routing element (switch or router) port.
- The switching / routing element examines the packet header and extracts the DLID / multicast GID to determine if it corresponds to a multicast group. An implementation may maintain this data as part of its internal route table, e.g. a bit-mask which corresponds to the output ports this packet should be forwarded.

- Switches or routers replicate the packet (implementation dependent) and forwards the packet onto the output port(s).

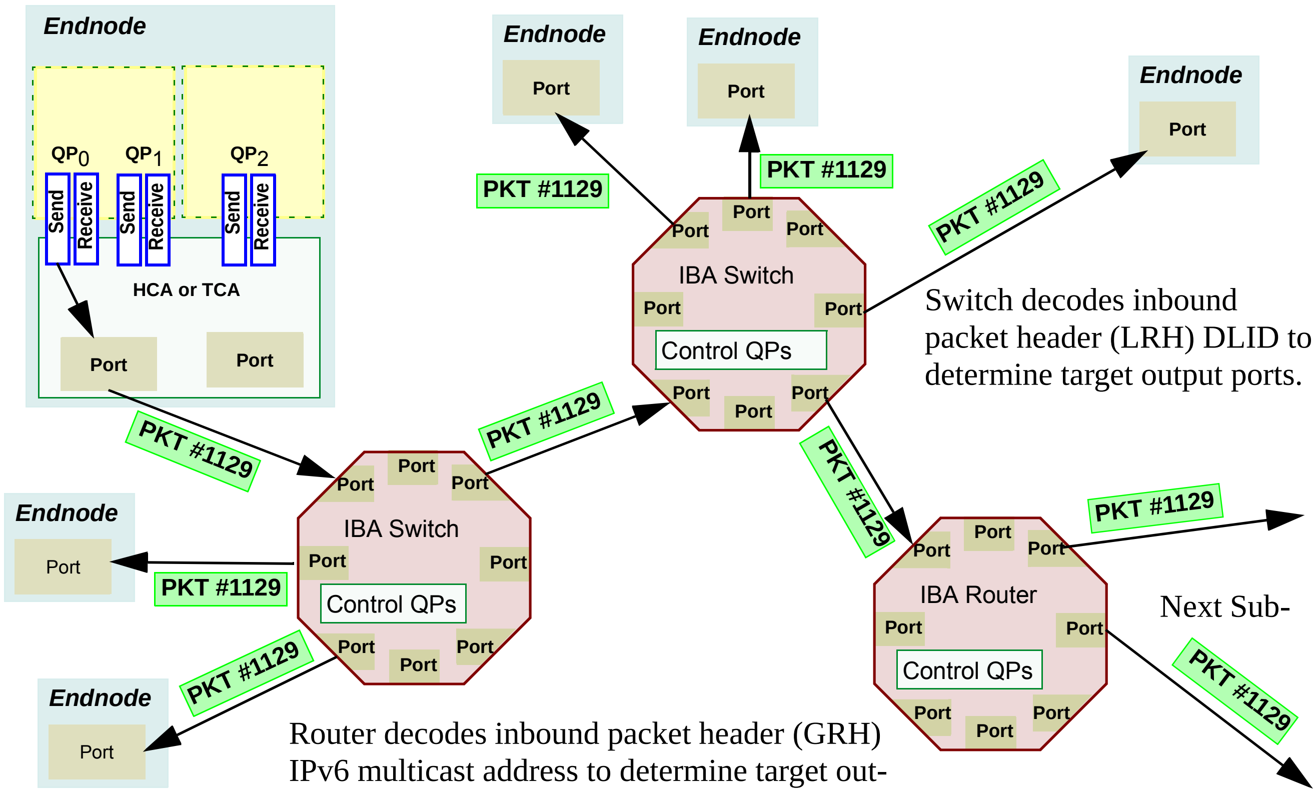


Figure 21 Example Unreliable Multicast Operation

3.5.11.2 GROUP MANAGEMENT

IBA V 1.0 does not define the multicast group management protocol to used to implement join and leave operations. However, the management interface and associated MADs to implement a multicast group protocol is specified. While these mechanisms are part of the Subnet Administration (SA), some actions are implicitly performed by the Subnet Manager (SM). For the following discussion, the term multicast management entity is used to describe the SA/SM expected responsibility with respect to multicast management. Refer to the Subnet Administration attributes of Multicast Group Record and Multicast Member Record for more information.

3.5.11.2.1 MULTICAST GROUP CREATE

The multicast group creation is an explicit operation in IBA, in order to provide a single control of group characteristics and allow members to join subversively. The group has to be created by the multicast management entity before a join can be successful:

1) An (administrative) application defines (or determines) a target multicast group address (GID). It specifies particular group characteristics

(PMTU, raw, P-Key, etc.) and creates the multicast group by invoking a multicast group create to the multicast management entity. This application may request a specific multicast LID or have one allocated for it.

2) The multicast management entity may notify appropriate routers on the subnet of the new group which is being created (not defined in IBA V 1.0). The router protocol should determine whether this multicast group is in operation within another subnet. If so the router returns the PMTU of the existing multicast group to determine whether the create is allowed or not.

3) The multicast management entity maps the multicast group address to an unused multicast LID or to the requested multicast LID.

3.5.11.2.2 MULTICAST GROUP JOIN

The multicast group join algorithm (applies to IBA and raw multicast groups) is defined as follows:

1) Application defines or determines the target multicast group address and invokes a multicast join operation.

2) The underlying join implementation determines if the associated endnode is participating in the multicast group. If it is, the application establishes a new local QP and performs the steps required to join this group. If not, the application invokes the management interface to communicate with the multicast group management entity.

3) The multicast management entity performs the following steps upon receiving a join request:

   a) Validate the multicast group address - fail join operation if invalid.

   b) Validate the requested PMTU - fail join operation if invalid.

   c) Verify the switch attached to the endnode is capable of multicast operation. The switch either supports multicast operation via packet replication or it can be configured to send all multicast packets to the endnode-attached port.

   d) If the multicast group address is currently in operation within this subnet, take the following actions:

      i) Verify all switches and routers which are participating in this multicast group can support the requested PMTU. If they cannot, the join operation fails.

      ii) Each multicast group is implemented by defining a logical routing tree across the participating switches. Rebuild / modify the routing tree to include the new endnode. The multicast management entity informs fabric management to update the associated route forwarding tables within all switches and routers to reflect this new topology.

e) If the multicast group address is not operating within this subnet, take the following steps.

   i) Inform each router within this subnet of the join operation. The router protocol should determine whether this multicast group is in operation within another subnet. If so, the router returns the PMTU of the existing multicast group to determine whether the create and subsequent join operation is allowed or not.

   ii) Map the multicast group address to an unused multicast LID.

   iii) Establish a multicast routing tree and update the associated switch and router route forwarding tables accordingly.

   iv) Create the group and assign the PMTU to the multicast group.

   v) Return the multicast LID and associated group characteristics to the endnode and allow multicast operations to be initiated.

f) Each router within this subnet is informed of successful multicast join operation. Routers invoke the appropriate multicast group management operations to add this subnet as participating in the associated multicast group. This protocol is outside the IBA specification.

4) Add the member to the group.

3.5.11.2.3 MULTICAST GROUP LEAVE

When an application leaves a multicast group, the following algorithm is used:

1) The application's QP is removed as a target for the multicast group. If there are QPs still participating in this multicast group, no further action is required.

2) If there are no more QPs on this port participating within the multicast group, the leave implementation informs the multicast management entity that this endnode is no longer participating in this multicast group. The multicast management entity takes the following step:

   a) Update the switch and router route forwarding table(s) to effectively remove this endnode as a target for packets associated with this multicast group.

   b) Remove the member from the group.

3.5.11.2.4 MULTICAST GROUP DELETE

When an (administrative) application deems there is no need for a multicast group or there are no other endnodes participating in a multicast group, the a multicast group may be deleted. Upon receiving the delete request, the multicast management entity takes the following steps:

1) Unmap the multicast LID from the multicast group address.
2) Inform each router within this subnet that this subnet is no longer participating in the associated multicast group.

3.5.11.3 MULTICAST PRUNE

To improve fabric efficiency, the multicast group management entity should periodically verify that all endnodes and routers participating within a multicast group are still participating and if they are not, it should prune them from the multicast group by performing the multicast group leave algorithm. The verification period is outside the scope of IBA V1.0.

3.5.12 VERBS

IBA describes the service interface between a host channel adapter and the operating system by a set of semantics called *Verbs*. Verbs describe operations that take place between a host channel adapter and its operating system based on a particular queuing model for submitting work requests to the channel adapter and returning completion status.

The intent of Verbs is not to specify an API, but rather to describe the interface sufficiently permitting OS venders to define appropriate APIs that take advantage of the architecture.

Verbs describe the parameters necessary for configuring and managing the channel adapter, allocating (creating and destroying) queue pairs, configuring QP operation, posting work requests to the QP, getting completion status from the completion queue.

3.6 CHANNEL & MEMORY SEMANTICS

IBA communications provide the user with both channel semantics and memory semantics since both are useful for I/O and IPC. Channel semantics, sometimes called Send/Receive, refers to the communication style used in a classic I/O channel – one party pushes the data and the destination party determines the final destination of the data. The message transmitted on the wire only names the destination's QP, the message does not describe where in the destination consumer's memory space the message content will be written.

With memory semantics the initiating party directly reads or writes the virtual address space of a remote node. The remote party needs only communicate the location of the buffer; it is not involved with the actual transfer of the data.

A typical I/O transaction might use a combination of channel and memory semantics. For example, a host process might initiate an I/O operation by using channel semantics to send a disk write command to an I/O device. The I/O device examines the command and uses memory semantics to read the data buffer directly from the memory space of the processor node. After the operation is completed, the I/O unit then uses channel semantics to push an I/O completion message back to the processor node.

3.6.1 COMMUNICATION INTERFACE

"*Channel adapter*" is the term that identifies the hardware that connects a node to the IBA fabric (and includes any supporting software). The channel adapter for a processor node is called a "*host channel adapter*" (HCA) and a channel adapter in an I/O node is a "*target channel adapter*" (TCA). A consumer communicates through one or more "queue pairs" (QP). An HCA typically supports hundreds or thousands of QPs while a TCA might support less than ten QPs.

It is the QP that is the communication interface. The user initiates work requests (WR) that causes work items, called WQEs, to be placed onto the queues and the channel adapter executes the work item.

Specifically, the operations supported for Send Queues are:

- **Send Buffer** -- a channel semantic operation to push a local buffer to a remote QP's receive buffer. The Send WR includes a gather list to combine data from several virtually contiguous local buffer segments into a single message that is pushed to a remote QP's Receive Buffer. The local buffer's virtual addresses must be in the address space of the consumer that created the local QP.
- **RDMA Read** -- a memory semantic operation to read a virtually contiguous buffer on a remote node. The RDMA Read operation reads a virtually contiguous buffer on a remote endnode and writes the data to a local memory buffer.

  Like the Send operation, the local buffer must be in the address space of the consumer that created the local QP.

  The remote buffer must be in the address space of the remote consumer owning the remote QP targeted by the RDMA Read.
- **RDMA Write** -- a memory semantic operation to write a virtually contiguous buffer on a remote node. The WR contains a gather list of local buffer segments and the virtual address of the remote buffer into which the data from the local buffer segments are written.

  Like the Send WR, the local buffer must be in the address space of the consumer that created the local QP.

  The remote buffer must be in the address space of the remote consumer owning the remote QP targeted by the RDMA Write.
- **Atomic** -- a memory semantic operation to do an atomic operation on a remote 64 bit word. The Atomic operation is a combined Read, Modify, and Write operation.

An example of an Atomic operation is the Compare and Swap if Equal operation. The WR specifies a remote memory location, a compare value, and a new value. The remote QP reads the specified memory location, compares that value to the compare value supplied in the message, and only if those values are equal, then the QP writes the new value to that same memory location. In either case the remote QP returns the value it read from the memory location to the requesting QP. The other atomic operation is the FetchAdd operation where the remote QP reads the specified memory location, returns that value to the requesting QP, adds to that value a value supplied in the message, and then writes the result to that same memory location.

- **Memory Bind** -- a memory management operation that changes the binding of a memory window. The Bind Memory Window operation associates a previously allocated Memory Window to a specified address range within an existing Memory Region, along with a specified set of remote access privileges.

For Receive Queues, there is only a single type of WR:

- **Post Receive Buffer** -- a channel semantic operation describing a local buffer into which incoming Send messages are written. The WR includes a scatter list describing several local buffer segments. The contents of an incoming Send message is written to these buffer segments in the order specified. The buffer's virtual addresses must be in the address space of the consumer that created the local QP. A WR without a scatter-gather list may be used to receive Immediate Data from a Write or a zero length Send operation.

Zero processor copy data transfer, with no kernel involvement, is key in providing high bandwidth, low latency communication. A consumer can transfer a data buffer via the QP directly from where the buffer resides in memory. Furthermore, the protection provided by R_Keys & L_Keys (memory registration) removes the need for the OS to validate data transfers. Thus the OS may allow posting the WQE from user-mode, bypassing the operating system, and thus consuming fewer instruction cycles.

IBA operations support the use of virtual addresses and existing virtual memory protection mechanisms to assure correct and proper access to all memory. Thus IBA applications are not required to use physical addressing for any operation.

A consumer can use either of two mechanisms to enable remote access to its memory (RDMA). First, when registering its memory (creating a Memory Region), the consumer can simply enable remote access for the entire Memory Region. If more control of remote access is desired, the consumer can allocate a Memory Window and bind it to a subset of an existing Memory Region. Either approach results in an R_Key. The consumer then provides that R_Key and the virtual address of the data buffer to a remote node for use in subsequent RDMA operations. Only an incoming RDMA request with a correct R_Key can gain access to the specific area of memory. Furthermore, the QP and the Memory Region or Window must be in the same protection domain.

3.6.2 IBA Transport Services

The IBA transport mechanisms provide multiple classes of communication services. When a QP is created, it is configured to provide one of these classes of transport services:

- **Reliable Connection** (acknowledged - connection oriented)
- **Reliable Datagram** (acknowledged - multiplexed)
- **Unreliable Connection** (unacknowledged - connection oriented)
- **Unreliable Datagram** (unacknowledged - connectionless)
- **Raw Datagram** (unacknowledged - connectionless)

The **Reliable Connection** service associates a local QP with one and only one remote QP. Thus a Send Buffer WQE placed on one QP causes data to be written into the Receive Buffer of the associated QP. RDMA operations operate on the address space of the associated QP.

A connected service requires each consumer to create a QP for each consumer with which it wishes to communicate. Thus if there are M consumers on each of N platforms that all wish to communicate via connected class of service, then each platform requires M^2 * N QPs. This assumes that each consumer on a particular platform communicates with consumers (including itself) on that same platform by taking advantage of the ability to connect a QP to a QP on the same node.

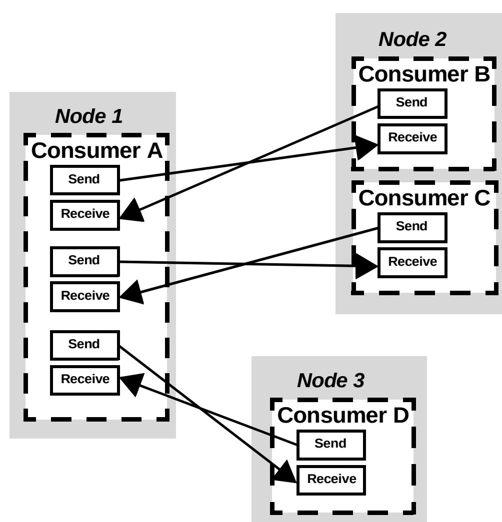


Figure 22 Connected Service

The Reliable Connection is reliable because the channel adapter can maintain sequence numbers and acknowledges all messages. A combination of hardware and channel adapter software retries any failed communications. The consumer of the QP sees reliable communications even in the presence of bit errors, receive buffer under runs, network congestion, and if alternate paths exist in the fabric, failures of fabric switches or links.

The acknowledgments ensure data is delivered reliably between the associated QPs and thus between each node’s memory.

The acknowledgment is not a consumer level acknowledgment -- it doesn’t validate that the receiving consumer has consumed the data. The acknowledgment only means the data has reached the destination.

The **Unreliable Connection** service also associates a local QP with one and only one remote QP. Thus a Send Buffer request placed on one QP causes data to be written into the Receive Buffer of the associated QP. RDMA Write operations operate on the address space of the associated QP.

Unlike reliable connection service, unreliable connection does not acknowledge and thus does not have the ability to resend lost or corrupted messages. Rather, lost or corrupted messages are simply dropped. Since there is no acknowledgment, RDMA Reads and Atomic operations are not supported. Because packets of an RDMA Write might be lost or corrupted, partial writing of a buffer might take place, but once a missing or corrupted packet is received, the write operation ceases until the start of a new message.

The **Unreliable Datagram** service is connectionless and unacknowledged. It allows the consumer of the QP to communicate with any unreliable datagram QP on any node. Receive operation allows incoming messages from any unreliable datagram QP on any node.

The Unreliable Datagram service greatly improves the scalability of IBA. Since the service is connectionless, an endnode with a fixed number of QPs can communicate with far more consumers and platforms compared to the number possible using the Reliable Connection and Unreliable Connection service.

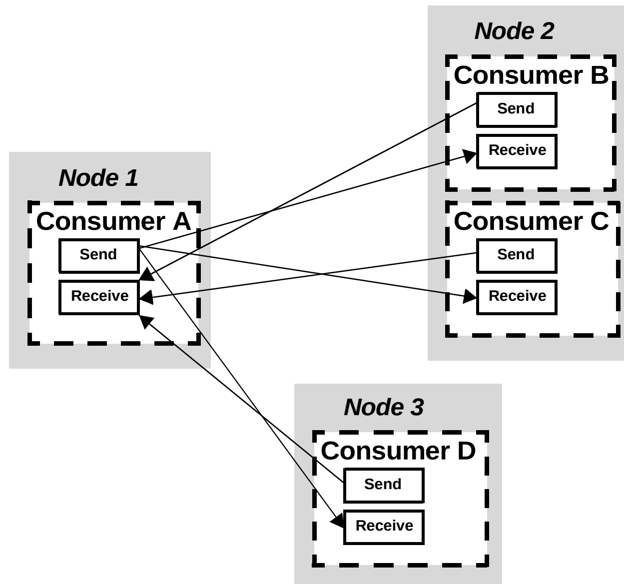


Figure 23 Datagram Service

This is the class of service used by management to discover and integrate new IBA switches and endnodes. It does not provide the reliability guarantees of the other service classes, but operates with less state maintained at each endnode. Unlike other services where messages are conveyed by multiple packets, the maximum message length is constrained in size to fit in a single packet.

The **Reliable Datagram** (RD) service is multiplexed over connections between nodes called End-to-end contexts (EEC) which allows each RD QP to communicate with any RD QP on any node with an established EEC. Multiple QPs can use the same EEC and a single QP can use multiple EECs (one EEC for each remote node per reliable datagram domain).

A reliable datagram domains (RDD) determine which sets of RD QPs can access which sets of EECs. Some possible reasons for multiple RDDs are traffic in different partitions, different QoS characteristics, security, and performance.

The EEC uses sequence numbers and acknowledgments associated with each message to ensure the same degree of reliability as with the Reliable Connection service. Each channel adapter maintains end-to-end specific state for each node to keep track of the sequence numbers, acknowledgments, and time-out values. Each EEC is shared by all Reliable Datagram QPs for that RDD.

For reliable datagram service on a per RDD basis, each consumer needs only to create a single QP and the node creates an EE context for each platform with which it communicates. Thus if there are M consumers on each of N platforms that all wish to communicate via IBA reliable datagram service, then each platform requires M QPs and N end-to-end contexts.

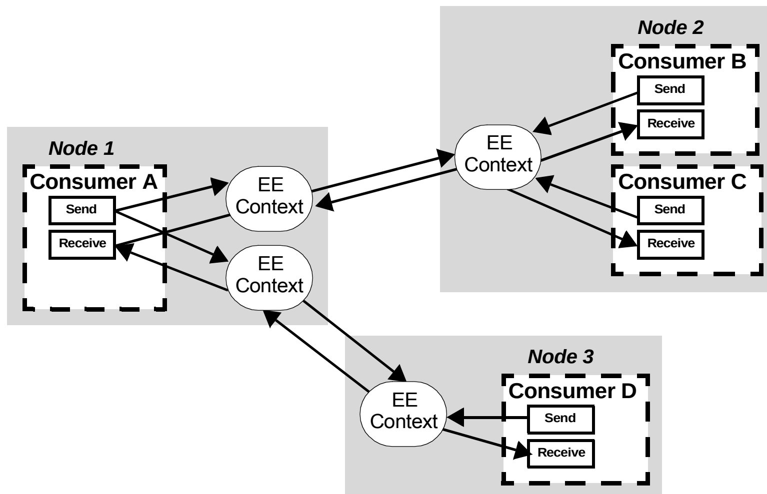


Figure 24 Reliable Datagram Service

The **Raw Datagram** service is not technically a transport but rather it is a data link service that allows a QP to send and receive raw datagram messages. There are two types of raw datagram service (EtherType and IPv6). The EtherType raw datagram packet contains a generic transport header that is not interpreted by the channel adapter, but it specifies the protocol type. The IPv6 raw datagram contains a global route header that identifies the protocol type.

Using IPv6 raw datagram service, the IBA channel adapter can support standard protocols layered atop IPv6, such as TCP and UDP. Thus native IPv6 packets can be bridged into the IBA SAN and delivered directly to a port and to its IPv6 raw datagram QP. This allows the raw datagram QP consumer to support multiple transport protocols.

Using EtherType raw datagram service, the IBA channel adapter can support standard protocols the same as Ethernet, including TCP and UDP as well as IPv4. Thus native ethernet packets can be bridged into the IBA subnet and delivered directly to a port and to its EtherType raw datagram QP.

When the QP is created, the consumer registers with the channel adapter in order to direct received datagrams to it (one QP for IPv6 and one for EtherType).

3.7 IBA LAYERED ARCHITECTURE

IBA operation can be described as a series of layers. The protocol of each layer is independent of the other layers. Each layer is dependent on the service of the layer below it and provides service to the layer above it.

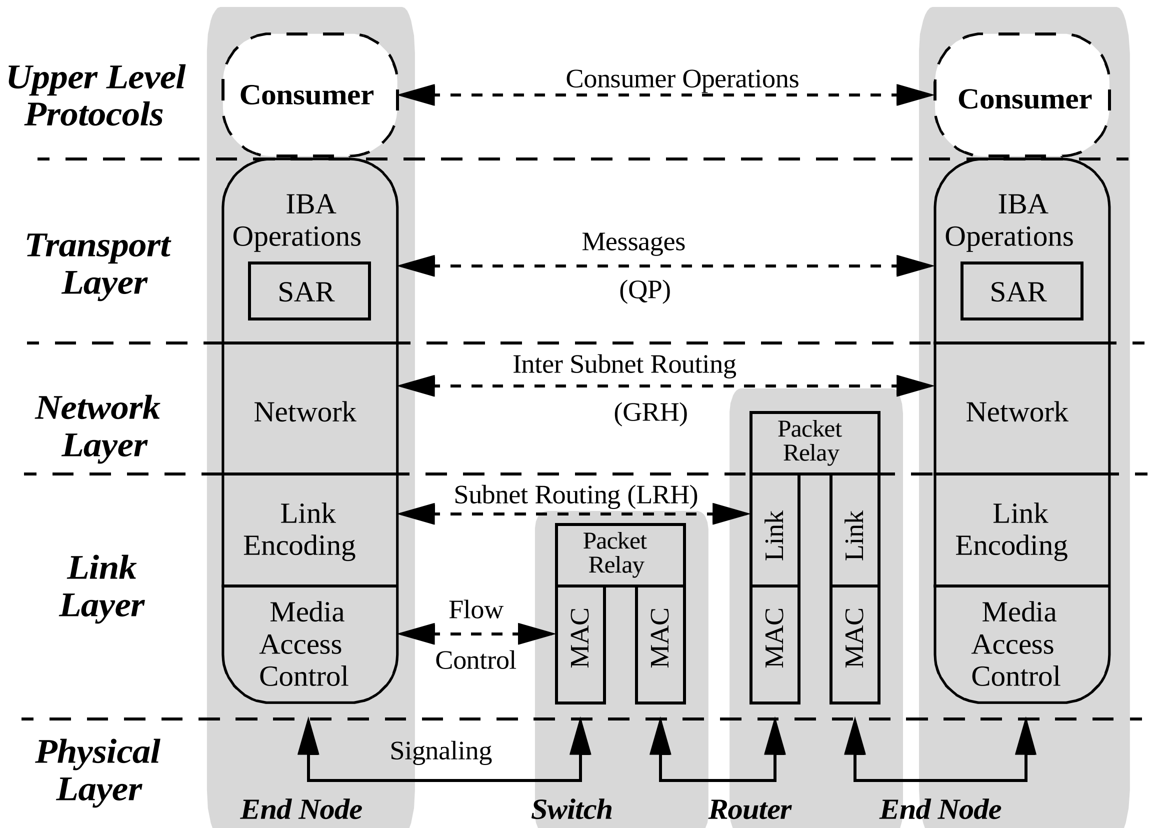


Figure 25 IBA Architecture Layers

3.7.1 PHYSICAL LAYER

The physical layer specifies how bits are placed on the wire to form symbols and defines the symbols used for framing (i.e., start of packet & end of packet), data symbols, and fill between packets (Idles). It specifies the signaling protocol as to what constitutes a validly formed packet (i.e., symbol encoding, proper alignment of framing symbols, no invalid or non-data symbols between start and end delimeters, no disparity errors, synchronization method, etc.).

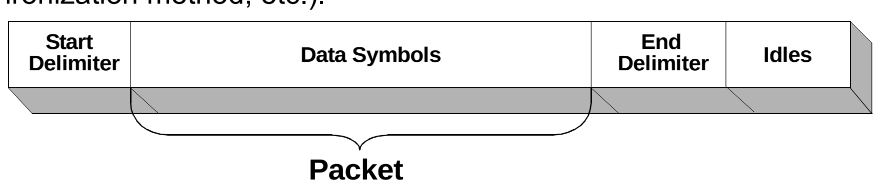


Figure 26 IBA Packet Framing

The physical layer specification is in Volume 2. It specifies the bit rates, media, connectors, signaling techniques, etc.

3.7.2 LINK LAYER

The link layer describes the packet format and protocols for packet operation, e.g. flow control and how packets are routed within a subnet between the source and destination. There are two types of packets.

- **Link Management Packet -** these are packets used to train and maintain link operation. These packets are created and consumed within the Link Layer and are not subject to flow control. Link management packets are used to negotiate operational parameters between the ports at each end of the link such as bit rate, link width, etc. They are also used to convey flow control credits and maintain link integrity. Link management packets are never forwarded to other links.
- **Data Packet -** these are the packets that convey IBA operations and they consist of a number of different headers, which might or might not be present.

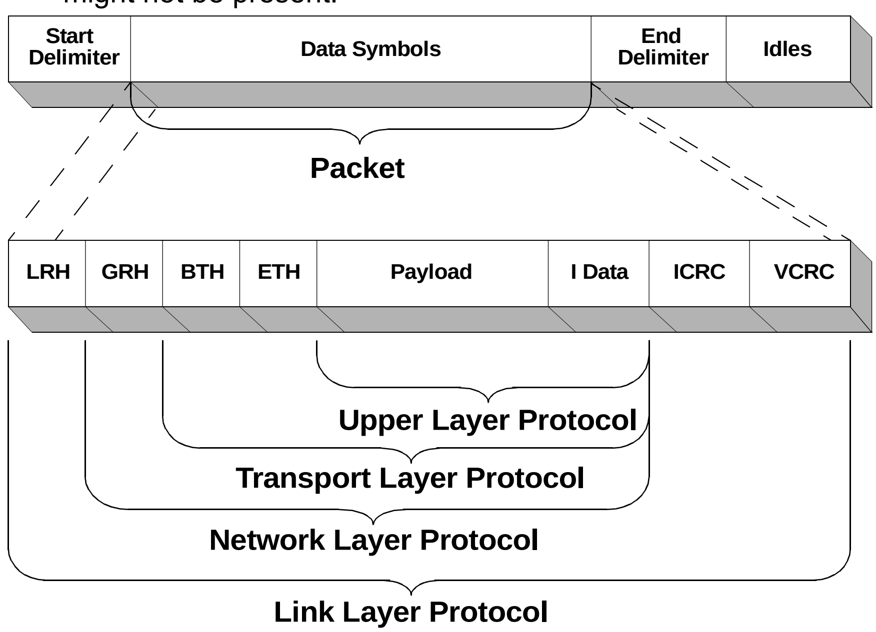


Figure 27 IBA Data Packet Format

The **Local Route Header** (LRH) is always present and it identifies the local source and local destination ports where switches will route the packet and also specifies the *Service Level* (SL) and VL on which the packet travels. The VL is changed as the packet traverses the subnet but the other fields remain unchanged.

The subnet manager assigns unique LIDs to each port of a channel adapter as well as the management entity of a switch. The source places the LID of the destination in the LRH and switches route the packet to that destination. If the packet is to be routed to another subnet, the packet's destination LID contains the LID of a router, otherwise the packet's destination LID specifies a LID assigned to a channel adapter (or switch, for certain of management packets).

There are two CRCs in each packet. The **Invariant CRC** (ICRC) covers all fields which should not change as the packet traverses the fabric. The **Variant CRC** (VCRC) covers all of the fields of the packet. The combination of the two CRCs allow switches and routers to modify appropriate fields and still maintain an end to end data integrity for the transport control and data portion of the packet. The coverage of the ICRC is different depending on whether the packet is routed to another subnet (i.e. contains a global route header).

Link level flow control is a credit based method where the receiver on each link sends credits to the transmitter on the other end of the link. Credits are per VL and indicate the number of data packets that the receiver can accept on that VL. The transmitter does not send data packets unless the receiver indicates it has room. VL15 (the management VL) is not subject to flow control.

3.7.3 NETWORK LAYER

The network layer describes the protocol for routing a packet between subnets.

The **Global Route Header** (GRH) is present in a packet that traverses multiple subnets. The GRH identifies the source and destination ports using GID in the format of an IPv6 address. Routers forward the packet based on the content of the GRH. As the packet traverses different subnets, the routers modify the content of the GRH and replace the LRH. But the source and destination GIDs do not change and are protected by the ICRC field. Routers recalculate the VCRC but not the ICRC. This preserves end to end transport integrity.

Each subnet has a unique subnet ID, the Subnet Prefix. When combined with a Port GUID, this combination becomes a port's Global ID (GID). A node might have other locally administrated Global IDs. The source places the GID of the destination in the GRH and the LID of the router in the LRH. Each router forwards the packet through the next subnet to another router until the packet reaches the target subnet. The last router replaces the LRH using the LID of the destination.

3.7.4 TRANSPORT LAYER

The network and link protocols deliver a packet to the desired destination. The transport portion of the packet delivers the packet to the proper QP and instructs the QP how to process the packet's data.

The transport layer is responsible for segmenting an operation into multiple packets when the message's data payload is greater than the *maximum transfer unit* (MTU) of the path. The QP on the receiving end reassembles the data into the specified data buffer in its memory.

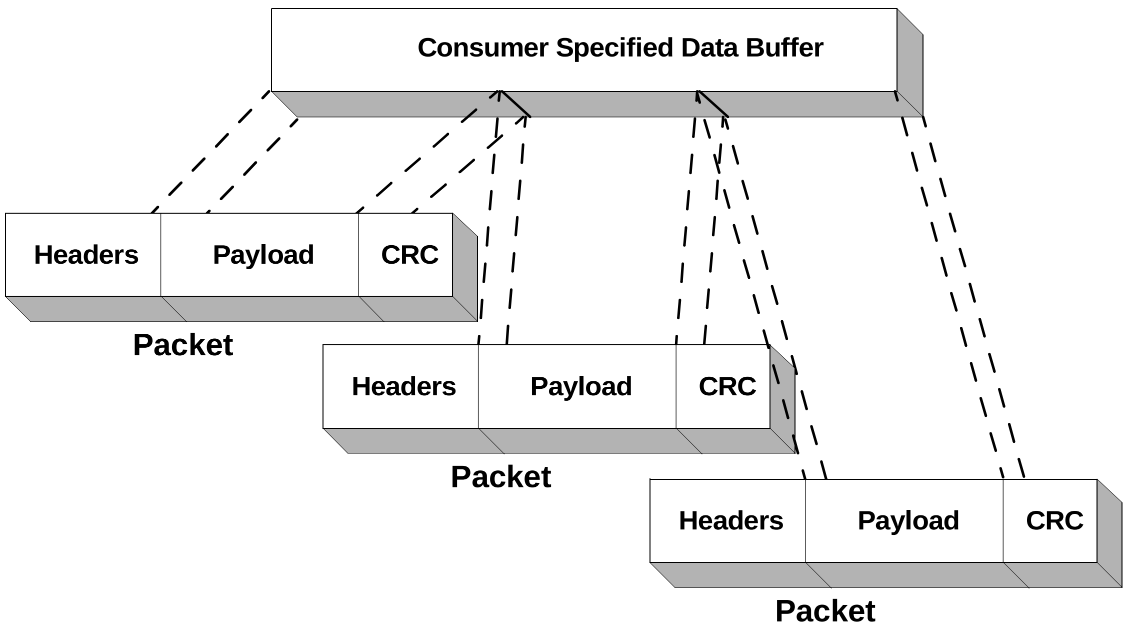


Figure 28 Segmentation of Data

The **Base Transport Header** (BTH) is present in all packets except for RAW datagrams. It specifies the destination QP and indicates the operation code, packet sequence number, and partition.

The operation code identifies if the packet is the first, last, intermediate, or only packet of a message and specifies the operation (Send, RDMA Write, Read, Atomic).

The packet sequence number (PSN) is initialized as part of the communications establishment process and increments each time the QP creates a new packet. The receiving QP tracks the received PSN to determine if it lost a packet. For reliable service, the receiver sends an ACK or NAK packet back to notify the sender that packets were or were not received correctly. In this case the recipient discards subsequent packets until the sender resends the missing messages. For unacknowledged service, when the recipient detects a missing packet, it aborts the current operation and discards all subsequent packets until it receives one that specifies a first or only operation code. Then operation continues.

There are various **Extended Transport Headers** (ETH) conditionally present depending on the class of service and the operation code.

For reliable datagram service, the ETH identifies the **EE context** that the QP uses to detect missing packets.

The first message of an RDMA Read or Write operation contains an **RDMA ETH** that specifies the virtual address, R_Key, and total length of the data buffer to read or write. Subsequent RDMA write packets provide the remainder of the data. The QP validates that the memory is properly registered for access by that QP and that the total data written does not overrun the length specified. For an RDMA Read operation, the QP fetches the data, segments it into Read Response packets and sends them to the originator. When receiving a RDMA response, the QP writes the data into the buffer specified in the WQE of the RDMA Read Request.

An Atomic operation contains an **Atomic ETH** that specifies the virtual address and R_Key of the memory location that is the object of the operation as well as 2 operands. The QP validates that the memory is properly registered for access by that QP. The QP fetches the data, returns that value to the originator, performs the operation, and writes the result back to memory. For the Compare & Swap operation, the QP compares the content of the memory location with the first operand, and if they match, then it writes the second operand to that same location. Otherwise it does not modify it. For the Fetch & Add operation, the QP performs an unsigned add using the 64-bit Add Data field in the Atomic ETH, and writes the result back to the same memory location. In either case, operation is atomic such that another QP is not allowed to modify that memory location between the time of the read and the subsequent write.

The **Immediate Data** (IMMDT) field is optionally present in RDMA WRITE and SEND messages. It contains data that the consumer placed in the Send or RDMA Write request and the receiving QP will place that value in the current receive WQE. An RDMA Write with immediate data will consume a receive WQE even though the QP did not place any data into the receive buffer since the IMMDT is placed in a CQE that references the receive WQE and indicates that the WQE has completed.

For reliable connection service, IBA defines an end-to-end message level flow control. This allows the receiver to send credits to the transmitter as WQEs are posted to the receive queue. The QP tracks the number of WQEs posted and retired from the receive queue and keeps track of the number of messages received. It adds these numbers together to achieve a message limit value which it sends to the transmitter on the other end of the connection. The transmitter keeps track of the total number of messages that it creates and stops transmitting when it reaches the limit value established by the other end of the connection.

3.7.5 UPPER LAYER PROTOCOLS

As illustrated in Figure 29, IBA supports any number of upper layer protocols by various user consumers. IBA also defines messages and protocols for certain management functions. These management protocols are separated into Subnet Management and Subnet Services. Both of these have unique properties.

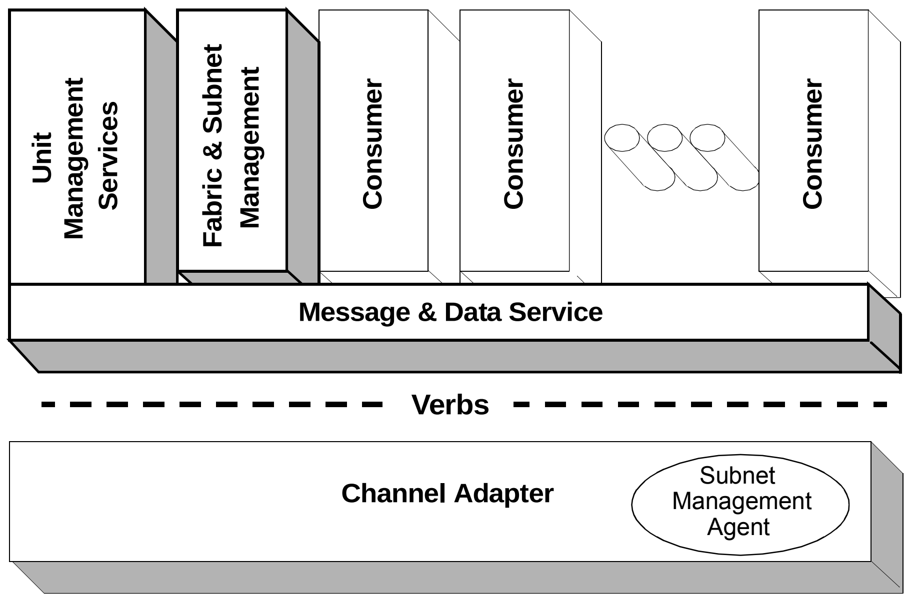


Figure 29 Upper Layers

3.7.5.1 SUBNET MANAGEMENT

Subnet Management is actually divided between the Subnet Manager (SM) application and the Subnet Management Agent (SMA). There only needs to be one subnet manager per subnet and it can reside in any node including switches and routers. Subnet management uses a special class of *Management Datagram* (MAD) called a *Subnet Management Packet* (SMP) which is directed to a special queue pair (QP0). As illustrated in Figure 30, each port has a QP0, and each node contains an SMA that:

- processes *Get()* and *Set()* SMPs received on QP0
- processes *Action()* SMPs received on QP0
- sends *GetResp()* SMPs out QP0
- sends *Trap()* SMPs out QP0.

A subnet manager:

- sends SMPs out QP0 to any port's QP0
- processes all SMPs received on QP0 except *Action()*, *Get(),* and *Set()* SMPs which are processed by that node's SMA.

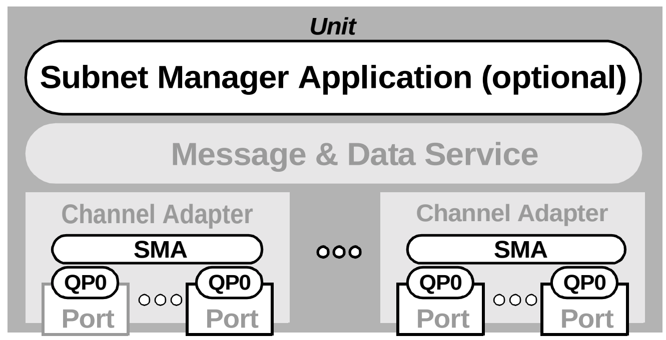


Figure 30 Subnet Management Elements

3.7.5.2 GENERAL SERVICES

General Service Agents (GSA*) actually consists of a number of management service agents as illustrated in Figure 31. Some of the services are optional. General services use a message format called a *General Management Packet* (GMP) which is a *Management Datagram* (MAD) and is normally directed to a special queue pair (QP1) called the General Service Interface (GSI). As illustrated in Figure 31, each port has a QP1, and all GMPs received on QP1 are processed by the one of the GSAs. The GSA is actually able to redirect GMPs for its particular class of service to another queue pair, allowing each GSA to maintain its own communication interface.

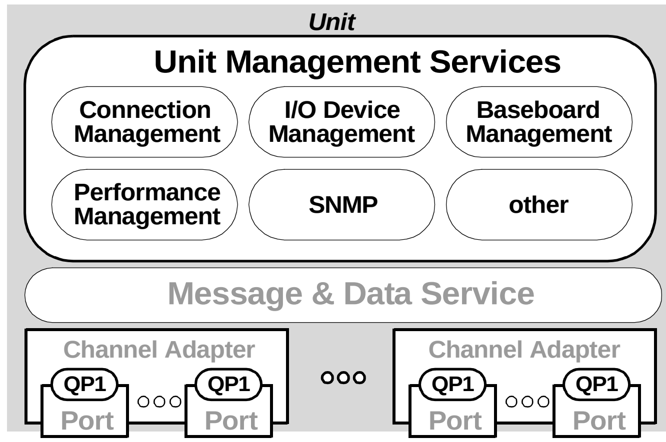


Figure 31 General Services

3.8 IBA TRANSACTION FLOW

A consumer interacts with an IBA channel adapter through a data structure called the Queue Pair, consisting of a Send Queue and a Receive Queue. A message is initiated by posting a work request which results in a WQE being placed on the Send Queue.

The channel adapter detects the WQE posting and accesses the WQE. The channel adapter interprets the command, validates the WQE's virtual addresses, translates it to physical addresses, and accesses the data. The outgoing message buffer is split into one or more packets. To each packet the channel adapter adds a transport header (sequence numbers, opcode, etc.). If the destination resides on a remote subnet the channel adapter adds a network header (source & destination GIDs). The channel adapter then adds the local route header and calculates both the variant and invariant checksums.

The flow of packets is subject to the link-level protocol over each link.

A packet is the unit of information that is routed through the IBA fabric. The packet is an endnode-to-endnode construct, in that it is created and consumed by endnodes. As the packet passes through switches, the switch may need to change the virtual lane and thus must replace the variant CRC with a new value but it does not touch the invariant CRC. If the packet passes through a router, the router changes the local route header and updates fields in the global route header, again updating the variant CRC but not changing the invariant CRC. Each switch and router moves the packet closer to its ultimate destination.

When a packet arrives at its final destination it goes through normal validity checks (e.g., framing violations, disparity, illegal characters, alignment, etc.) and both VCRC and ICRC are checked for integrity. The transport header identifies the target QP and the channel adapter uses context from that QP to validate that the packet came from the correct source, etc. and checks that the packet sequence number is valid (no missed packets). For a Send operation, the QP retrieves the address of the receive buffer from the next WQE on its receive queue, translates it to physical addresses, and accesses memory writing the data. If this is not the last packet of the message, the QP saves the current write location in its context and waits for the next packet at which time it continues writing the receive buffer until it receives a packet that indicates it is the last packet of the operation. It then updates the receive WQE, retires it, and sends an acknowledge message to the originator.

For reliable service types, if the QP detects one or more missing packets, it sends a NAK message to the originator indicating its next expected sequence number. The originator can then resend starting with the expected packet.

When the originator receives an acknowledgment, it creates a CQE on the CQ and retires the WQE from the send queue.

A QP can have multiple outstanding messages at any one time but the target always acknowledges in the order sent, thus WQEs are retired in the order that they are posted.

3.9 IBA Management Infrastructure

IBA management defines a common management infrastructure for

- Subnet Management - provides methods for a subnet manager to discover and configure IBA devices and manage the fabric.
- General management services
  - Subnet administration - provides nodes with information gathered by the SM and provides a registrar for nodes to register general services they provide.
  - Communication establishment & connection management between endnodes
  - Mechanisms to discover and manage I/O devices “behind” channel adapters
  - Configuration management - an authority for assigning I/O resources to hosts
  - Performance management - monitors and reports well-defined performance counters

- Baseboard management - provides for power & chassis management using IB-ML as defined in Volume 2
- SNMP Tunneling (SNMP) - provides method for sending and receiving information between management agents and management applications. This includes Simple Network Management Protocol (SNMP), Desktop Management Interface (DMI), and Common Information Model (CIM), as well as other standard and proprietary interfaces.

The subnet management physical and logical models are illustrated in Figure 32. The general service models are illustrated in Figure 33 and Figure 34.

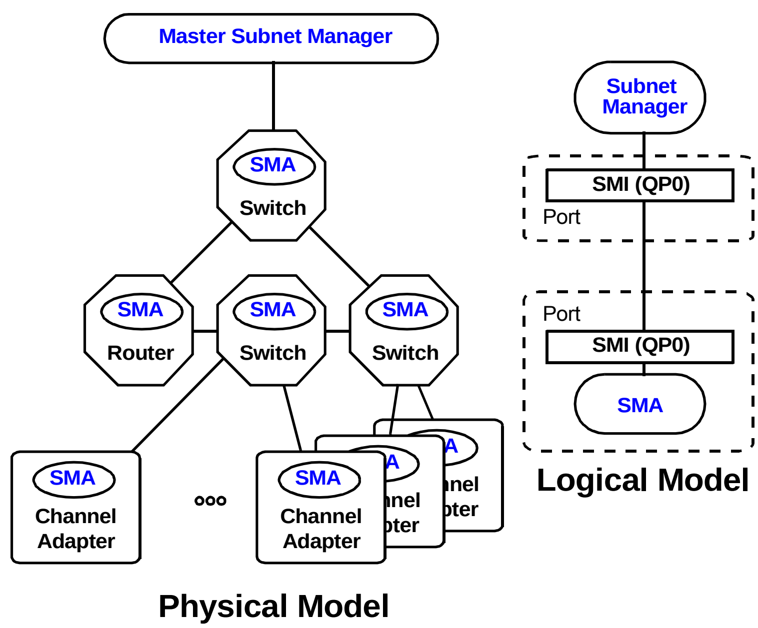


Figure 32 Subnet Management Models

Every channel adapter, switch, and router has a Subnet Management Agent (SMA) that responds to subnet management packets. Communication between the SM and SMAs use a well-known interface called the Subnet Management Interface (SMI) where each port has a QP with QP Number 0 (QP0) that is dedicated to sending and receiving SMPs.

Protection - The subnet manager can place a key (M_Key) in each node which can not be read by other nodes and prevents nodes without the M_Key from modifying a node's configuration. The SM only shares the M_Key with trusted peers as necessary. IBA also provides a lease expiration mechanism such that if the SM dies before is shares M_Key information with a successor, the lease expires, and the node returns to a state that allows the successor SM to establish a new M_Key.

IBA management defines the underlying interfaces and principles that allow IBA devices and the corresponding fabric to be discovered, initialized, and controlled. It defines a common management model and framework applicable to IBA-managed elements, identifies those elements, and defines their managed features. Management applications use this infrastructure to manage the IBA devices and communicate with other management applications.

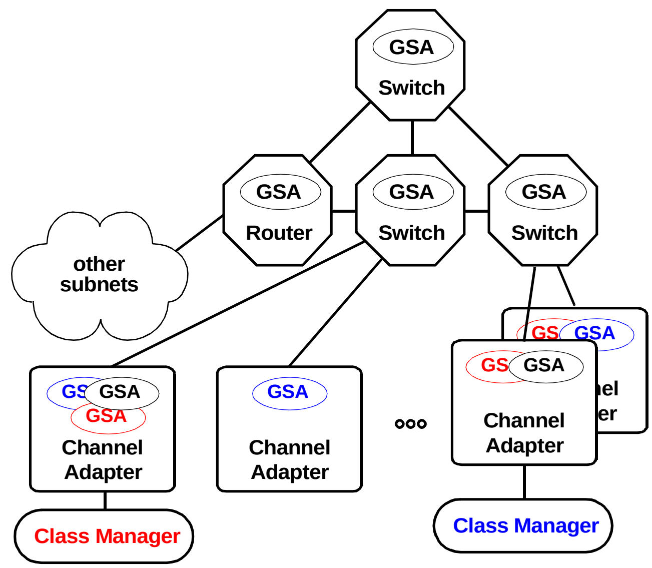


Figure 33 General Services Physical Model

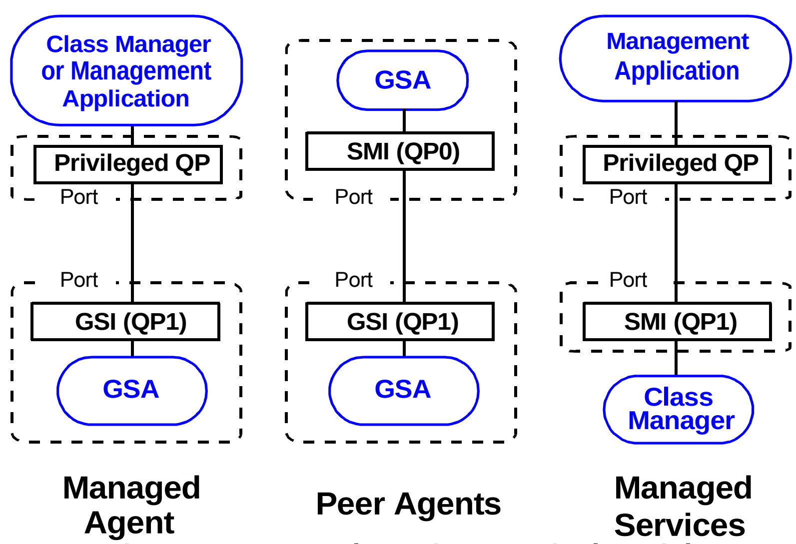


Figure 34 General Services Logical Models

IBA management infrastructure supports a number of different management service classes and logically provides for any node to host a class manager. Figure 34 illustrates different ways that management classes can use the management infrastructure.

- The Managed Agent model allows a class manger or management application to manage nodes through a General Service Agent (GSA) defined for that class present on each node to be managed. This is the same model used for subnet management and is the model used for I/O device management, baseboard management, SNMP, and performance management classes.
- The Peer Agents model allows managers resident on each node as a GSA to communicate with each other. This is the model used for communication management class.
- The Managed Service model allows management applications to access class managers. This is the model used for subnet administration and I/O resource management classes.

IBA management entails a variety of concepts, including:

- A means of configuring and gathering information from endnodes, switches and routers.
- A diagnostics framework as a common error handling mechanism.
- Installation and configuration services to allow for discovery and initialization of the fabric and endnodes.
- A standard management packet called a “Management Datagram” or “MAD”.
- Subnet Management Packets (SMP) as a subset of the MADs to allow set and get operations specifically between the Subnet Manager and IBA devices.
- General Management Packets (GMP) as the remaining subset of the MADs that allow management operations between the Subnet Manager and IBA devices and management operations between IBA devices themselves.
- Communication management services to allow setup and teardown of communications between channel adapters.
- Partitioning services to configure ports of an endnodes to be members of one or more possibly overlapping sets called partitions.

IBA provides the means for the operating system to restrict access to the management infrastructure. For the SMI, subnet management packets must be sourced from QP0. The GSI uses a privileged Q_Key (i.e., a Q_Key with the most significant bit set). Host channel adapters do not permit a privileged Q_Key to be specified in a work request, rather the QP must be configured for privileged operation by configuring the QP context with the privileged Q_Key. This permits management applications and class mangers to maintain their own QPs. The GSI uses QP1 for initial communication but allows traffic for a particular class to be redirected to a privileged QP.

3.9.1 MANAGEMENT INTERFACES

IBA defines two well known QPs for management interfaces. **QP0** is reserved for subnet management and **QP1** is designated for general management services.

3.9.2 SUBNET MANAGEMENT INTERFACE

Every IBA port has a QP dedicated to subnet management. This is QP0. QP0 has special features that make it unique compared to other QPs.

- QP0 is permanently configured for Unreliable Datagram class of service.
- Each port of an IBT device has a QP0 that sends and receives packets.
- QP0 is a member of all partitions (i.e., can accept any packet
- specifying any partition)
- Only subnet management packets (SMPs) are valid
- Traffic for QP0 (i.e., SMPs) exclusively uses VL15, which is not subject to link-level flow control.

3.9.2.1 FABRIC INITIALIZATION

The subnet manager uses this service interface to poll and configure the fabric. Switches support a special routing mode known as *directed routing* that allows SMPs to be routed through switches prior to switches being configured with their forwarding database and prior to nodes being assigned local IDs. The subnet manager walks its way through the fabric sending SMPs to a device and discovering if it is a switch. Using directed routing, it can then send SMPs out each of the switch's ports to discover the devices connected to the switch. This process continues until the subnet manger discovers all of the devices and how they are interconnected.

Once the SM learns the subnet's topology, it configures each node with local IDs and configures the routing tables of switches. Once the fabric has been configured, SMPs can be sent using destination routing.

IBA allows multiple subnet mangers per subnet but only one can be the master manager. Thus IBA defines how a subnet manager detects the presence of another subnet manager and the arbitration mechanism for selecting which will be the master subnet manager.

3.9.2.2 GETS & SETS

Gets and Sets are the most common use of SMPs. The SM polls the fabric and learns its topology by sending SMP Get Request messages. Each destination responds by sending a SMP Get Response message that includes the requested data. The SM configures IBT devices by sending

SMP Set Request messages. This is effectively a Set & Get request. Each destination responds by sending a SMP Get Response message that includes the data values after the set action. Since not all parameters are settable or they might have limits, the originator inspects the response message to determine the true effect of the set request message.

3.9.2.3 TRAPS AND NOTICES

A trap is a message sent by a management agent to its class manger when certain asynchronous management events occur (such as protocol violations). Notices are a list of management events that are queued at the managed node and may be retrieved and cleared by the class manager or management application.

IBT devices use SMPs to send traps to the subnet manager when certain events occur. One such use is for a switch to send a trap to the subnet manager when it detects a state change on one of its ports (i.e., a topology change and/or device joining or leaving). Of course since SMPs are unreliable, the SM can not solely depend on this type of notification, but successful traps will decrease the latency in managing the topology change.

3.9.2.4 ACTIONS

The SM also uses SMPs to send action commands to devices. Wake is one such action that, if supported, causes a node to power up and become operational. Another action is the node reset command.

3.9.2.5 DIRECTED ROUTES

A SMP can specify the route it takes through the fabric. This is done by including in the SMP a list of port numbers that define a path through the subnet (i.e., the path vector). The path vector specifies the output port for each switch along the path. The packet contains two path vectors (one for the forward route and one for the reverse route), a direction bit that indicates which path vector to traverse, and a hop pointer that indicates the current position in the path vector. The reverse path vector is built by switches as they process the forward path vector.

When a switch receives a directed routed SMP, it uses the current hop pointer to identify where in the path vector it is. If the direction is “forward” it determines the output port from the forward path vector, updates the reverse path vector by adding the port number on which it received the SMP, increments the current hop pointer, and then forwards the packet out the specified output port. When the packet reaches the destination, the target device uses the reverse route field for the reply by simply changing the sense of the direction bit and sending the reply SMP out the port on which it was received. Because the direction is “reverse” each switch now decrements the current hop pointer, uses it to determine the original input port, and then sends the packet out that port.

3.9.3 General Service Interface

Every IBA channel adapter has a QP dedicated to general fabric services. This is QP1. QP1 has special features that make it unique compared to other QPs.

- QP1 is permanently configured for Unreliable Datagram class of service.
- Each port of an IBT device has a QP1 that sends and receives packets.
- QP1 is a member of all of the port's partitions (i.e., can accept any packet specifying a P_Key contained in the port's P_Key table).
- Only management datagrams (MADs) are valid
- Traffic for QP1 does not use VL15

3.9.3.1 Management Datagrams

IBA defines a standard format for management messages which supports common processing. Each MAD contains the same header format that identifies the class of management message and the method. SMPs are one class of management message, another is directed route SMPs. Other classes are called General Management Packets (GMPs) and include subnet administration, communication management, performance management, SNMP, device management, baseboard management.

IBA defines common methods that may be adopted by any class. These include Get, Set, Trap, Notice, and Action. Of course each management class defines their own set of attributes. These methods are sufficient for many classes but IBA provides for class specific methods.

3.9.3.2 Redirection

QP1 being a well known interface has its advantages and disadvantages. One disadvantage is that all management classes go into the same queue which tends to bottleneck and promote head of line blocking. Thus IBA defines a mechanism that allows the channel adapter to redirect general service requests to other QPs.

When a channel adapter receives a GMP on QP1, it may respond with a redirect response indicating a new port and QP. The originator then resends the request to the new address and also uses that address for all subsequent requests for that same management class.

3.10 I/O Operation

IBA I/O architecture supports a range of I/O implementation from simple native devices to massive I/O subsystems. The model for an I/O unit is shown in Figure 35. An I/O unit is composed of a channel adapter and a number of I/O controllers. The channel adapter of an I/O unit is referred to as a Target Channel Adapter (TCA). A TCA has the same functionality as the HCA, but unlike the HCA, it is not necessarily designed for generic use, which means that it only needs to support the capabilities required by its controllers.

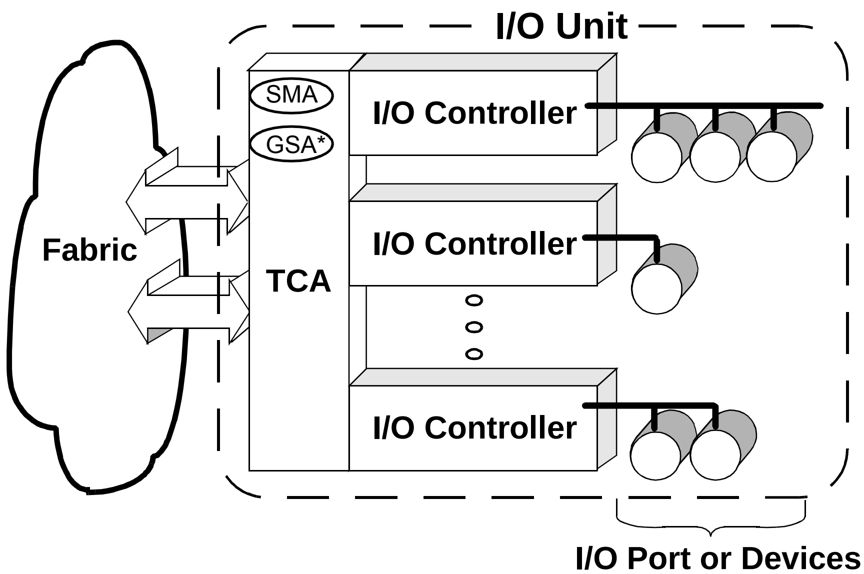


Figure 35 I/O Unit

I/O controllers represent the hardware and software that processes I/O transaction requests. Examples of I/O controllers are a SCSI interface controller, a RAID processor, a storage array processor, a LAN port controller, a disk drive controller, a console service.

The I/O unit contains a Subnet Management Agent (SMA) that responds to SMPs received on QP0. The I/O unit also contains general service agents (GSA*) that responds to GMPs received on the GSI (QP1). The GSA* contains at least Communication Management and I/O Device Management (DevMgt). Each I/O controller is registered with the DevMgt GSA such that it can respond to DevMgt GMPs with specific information about the controller.

Typically an I/O resource manager in the processor node sends DevMgt GMPs to an I/O unit to discover the attributes of the controllers. The attributes contain sufficient information for the I/O resource manager to identify the appropriate I/O driver. The I/O resource manager loads the driver, if necessary, and configures the I/O driver with the identity of the controller (IO Unit and Controller ID). The I/O driver then creates the appropriate communication ports (i.e., QPs) on the processor node and calls the processor node's communication manager to create the appropriate connections with the I/O controller. Once the connections are established, the I/O driver exchanges control messages and data over the connections.

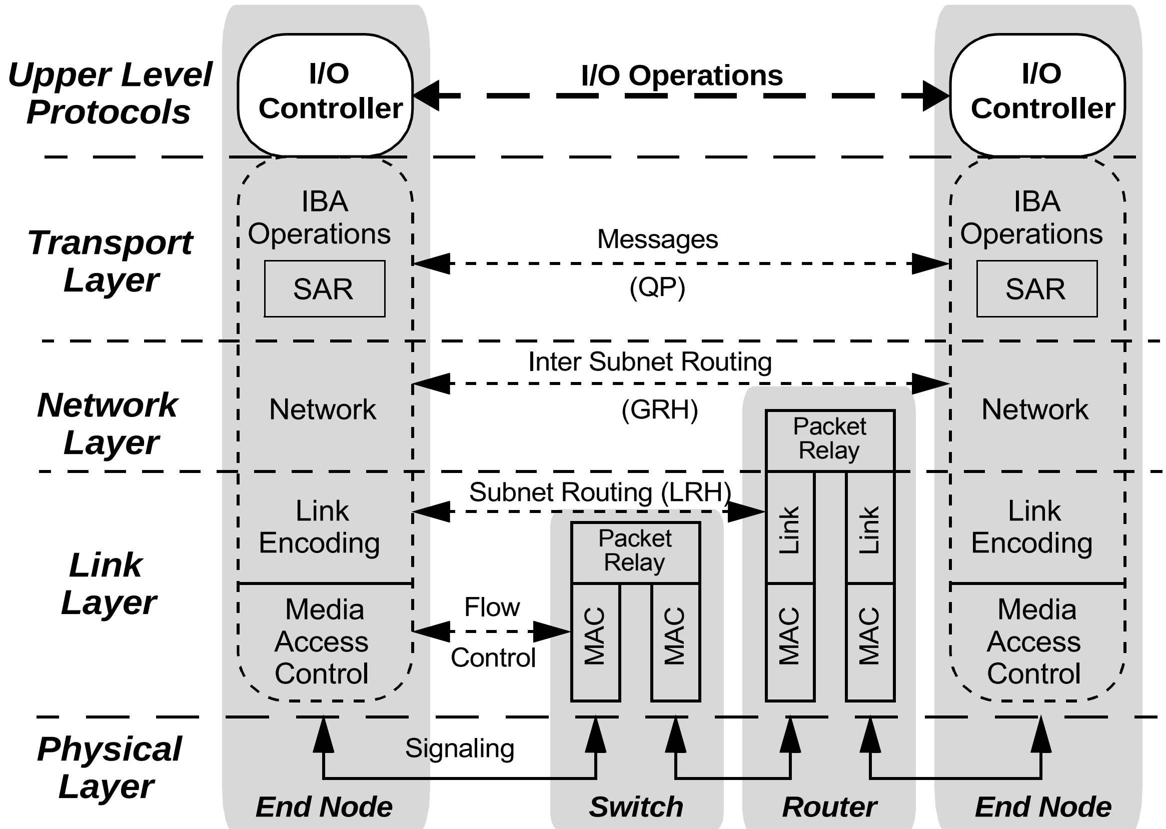


Figure 36 IO Operation

The number of communication ports used by the driver is an implementation variable. An I/O driver may use any available class of service (reliable connection, unreliable connection, reliable datagram, or unreliable datagram) and might use various classes of service for different communication ports.

CHAPTER 4: ADDRESSING

This chapter defines IBA addressing terminology and concepts. To facilitate understanding, refer to the following figures.

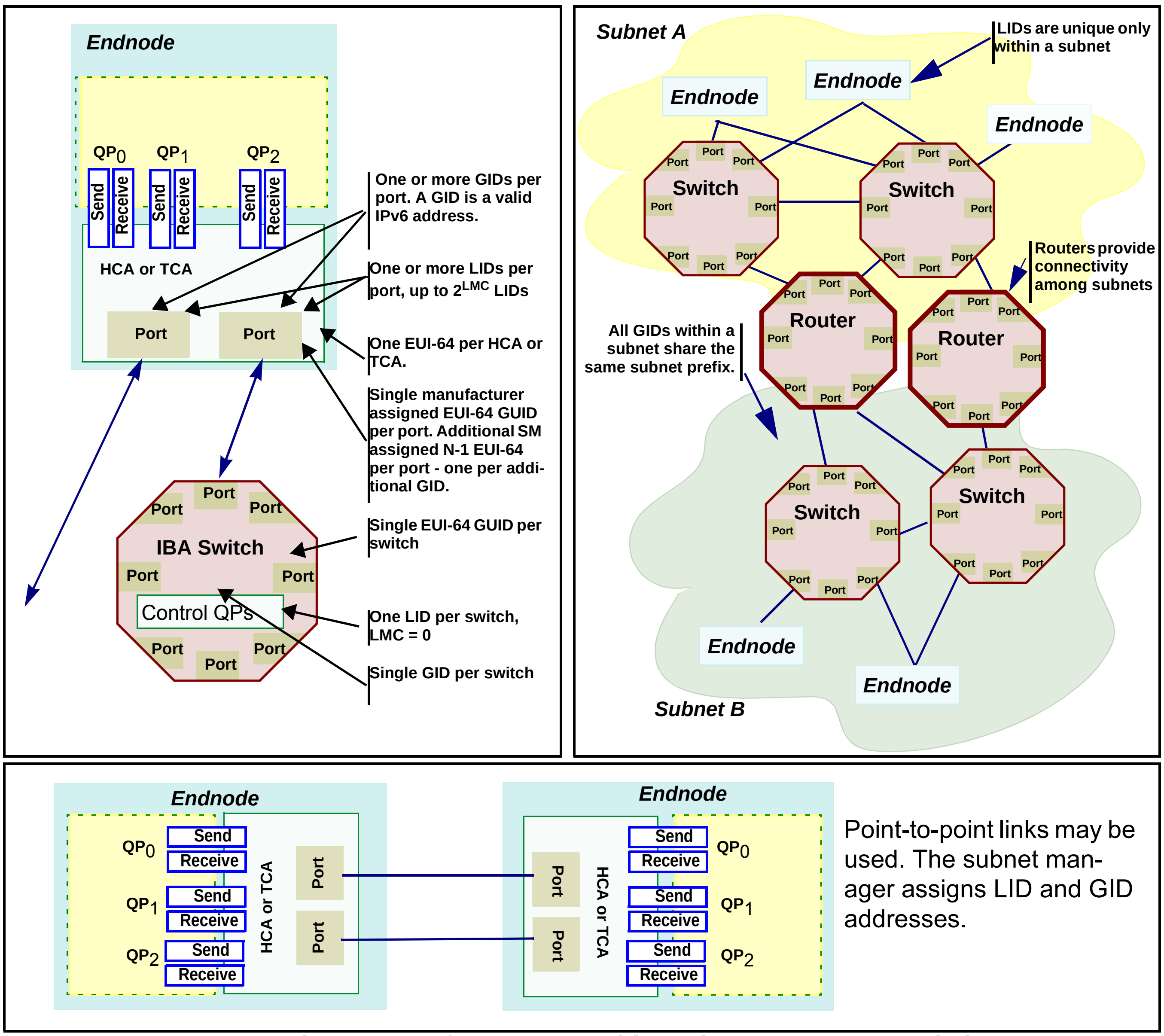


Figure 37 Reference IBA Address / Component Association

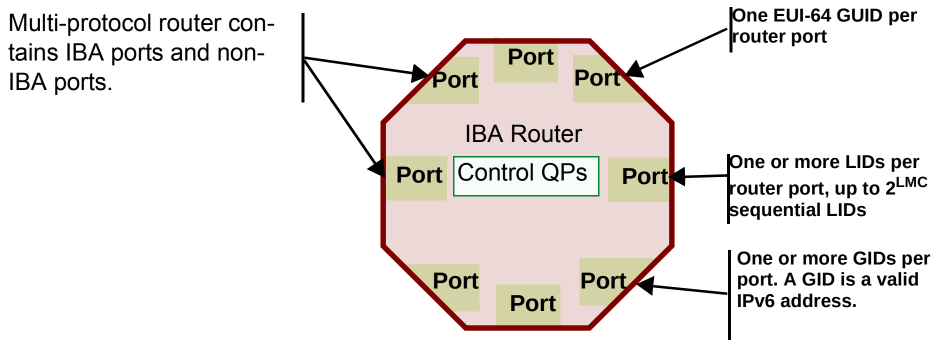


Figure 38 Reference IBA Router Address Association

4.1 TERMINOLOGY AND CONCEPTS

Unicast Identifier: An identifier for a single channel adapter or router port. A packet sent to an unicast identifier is delivered to the port identified by that identifier. IBA defines two unicast identifier - a global identifier (GID) - may be unique across subnets - and local identifier (LID) - unique only within a subnet).

Multicast Identifier: An identifier for a set of destination ports on channel adapters or routers. A packet sent to a multicast identifier is delivered to all ports identified by that identifier. IBA defines two multicast identifiers - a global identifier (GID) used by applications to address a multicast group and route packets between subnets and a local identifier (LID) used to switch packets within a subnet.

EUI-64: IEEE defined 64-bit identifier assigned to a device. The EUI-64 is a 64-bit identifier created by concatenating a 24-bit company_id value and a 40-bit extension identifier. The company_id is assigned by the IEEE Registration Authority; the extension identifier is assigned by the organization with the assigned company_id.

- The universal / local bit in IEEE EUI-64 shall be set to one to indicate global scope or set to zero to indicate local scope. The manufacturer assigns an EUI-64 with global scope set. A SM may assign additional EUI-64 with local scope indicated.
- For additional details, see: “Guidelines For 64-bit Global Identifier (EUI-64) Registration Authority”at www.standards.ieee.org/regauth/oui/tutorials/EUI64.html

GUID (Global Unique Identifier): A globally unique EUI-64 compliant identifier.

C4-1: Each HCA, TCA, switch and router shall be assigned an EUI-64 GUID by the manufacturer.

C4-2: Each port on a CA or router shall be assigned an EUI-64 GUID by the manufacturer.

Subnet Prefix: A 0 to 64-bit - as a function of scope - identifier used to uniquely identify a set of links, channel adapter ports, and switches which are managed by a common subnet manager.

GID Prefix: A 64-bit identifier (upper 64-bits of a GID) created by concatenating address scope bits, potentially a small number of “filler” bits, and potentially a subnet prefix - filler and subnet prefix presence is a function of the address scope.

GID (Global Identifier): A 128-bit unicast or multicast identifier used to identify a port on a channel adapter, a port on a router, a switch, or a multicast group. A GID is a valid 128-bit IPv6 address (per RFC 2373) with additional properties / restrictions defined within IBA to facilitate efficient discovery, communication, and routing. Note: These rules apply only to IBA operation and do not apply to raw IPv6 operation unless specifically called out.

C4-3: GIDs shall comply with the rules defined within 4.1.1 GID Usage and Properties on page 117:

4.1.1 GID Usage and Properties

1) Each port on a CA or router shall be assigned at least one unicast GID. The first unicast GID assigned shall be created using the manufacturer assigned EUI-64 identifier. This GID is referred to as **GID index 0**.
2) A unicast GID shall be created using one of the following mechanisms:
    a) Concatenation of the default GID prefix (0xFE80::0) and any of the CA or router port’s or the switch’s assigned EUI-64 identifier (at any GID index) - this is referred to as a default GID. A packet containing a GRH with a GID with this prefix must never be forwarded by a router, i.e. it is restricted to the local subnet.
    b) Concatenation of a subnet manager assigned 64-bit GID prefix and the CA or router port’s or the switch’s manufacturer assigned EUI-64 identifier. A subnet shall have only one assigned GID prefix at any given time (at GID index 0).
    c) Assignment of a GID by the subnet manager. The subnet manager creates a GID by concatenating the GID prefix with a set of locally assigned EUI-64 values (at GID index 1 or above).

    All CA and router ports and switches must be assigned at least one unicast GID using either (a) or (b). CA and router ports may be assigned additional unicast GIDs using (c).

3) Any QP in a CA, switch or router shall be addressable using the default GID prefix (0xFE80::0) in addition to the assigned GID for that QP. This allows a subnet to transition from a default GID prefix state to a managed state without interrupting existing communication sessions.

4) The maximum number (N) of unicast GIDs supported per CA or router port is implementation specific. The subnet manager may assign N-1 additional unicast GIDs. Each of these N-1 GIDs is created by concatenating one subnet manager assigned EUI-64 identifiers (the local bit set) with the GID prefix.

5) The unicast GID address 0:0:0:0:0:0:0:0 is reserved - referred to as the Reserved GID. It shall never be assigned to any channel adapter, switch, or router. It shall not be used as a destination address or in a global routing header (GRH).

6) The unicast GID address 0:0:0:0:0:0:0:1 is referred to as the loopback GID and is only used by raw IPv6 services - it is not used by IBA transport services. It shall never be assigned to a channel adapter or router nor be present in any IBA packets.

7) The unicast GID subnet prefix shall be limited to the upper 64-bits of the GID address space. The number of subnet prefix bits may further be limited by filler and scope bits - see below.

8) The lower 64-bits of the unicast GID cannot be further partitioned into subnets.

9) The lower 64-bits of a unicast GID shall be subnet unique. If the universal / local bit is set to universal, then the assignment must be globally unique.

10) The GRH (Global Route Header) shall contain valid source and destination GIDs. For raw IPv6 packets, an IPv6 routing header is with the source and destination addresses complying to RFC 2373.

11) Unicast GID scoping shall be:

   a) Link-local - A unicast GID used within a local subnet using the default GID prefix. Routers must not forward any packets with either link-local source or destination GIDs outside the local subnet. A link-local GID has the following format:

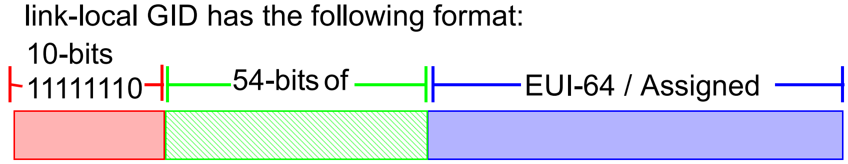


Figure 39 Link-Local Unicast GID Format

b) Site-local - A unicast GID used within a collection of subnets which is unique within that collection (e.g. a data center or campus) but is not necessarily globally unique. Routers must not forward any packets with either a site-local Source GID or a site-local Destination GID outside of the site.

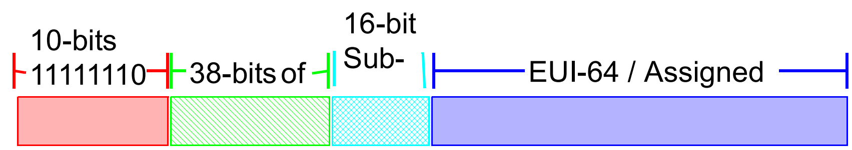


Figure 40 Site-Local Unicast GID Format

c) Global - A unicast GID with a global prefix, i.e. a router may use this GID to route packets throughout an enterprise or internet. The global GID format is:

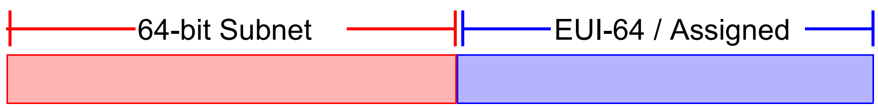


Figure 41 Unicast Global GID Format

12) A multicast GID is an identifier for a group of ports on channel adapters and routers. The multicast GID format is:

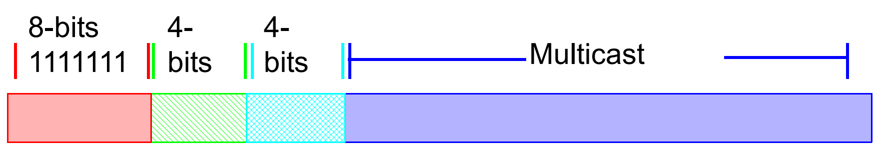


Figure 42 Multicast GID Format

a) 8-bits of 11111111 at the start of the GID identifies this as being a multicast GID.

b) Flags is a set of four 1-bit flags: 000T with three flags reserved and defined as zero ('0'). The T flag is defined as follows:

vi) T = 0 indicates this is a permanently assigned (i.e. well-known) multicast GID. See RFC 2373 and RFC 2375 as reference for these permanently assigned GIDs.

vii) T = 1 indicates this is a non-permanently assigned (i.e. transient) multicast GID.

c) Scope is a 4-bit multicast scope value used to limit the scope of the multicast group. The following table defines scope value and interpretation.

Table 3 Multicast Address Scope

Scope Value	Address Scope
0	Reserved
1	Unassigned
2	Link-local
3	Unassigned
4	Unassigned
5	Site-local
6	Unassigned
7	Unassigned
8	Organization-local
9	Unassigned
0xA	Unassigned
0xB	Unassigned
0xC	Unassigned
0xD	Unassigned
0xE	Global
0xF	Reserved

13) A CA or router may join zero, one or more multicast groups, i.e. a CA or router port may be assigned zero, one or more multicast GIDs.

14) Multicast GIDs shall not appear as the source GID in the GRH.

15) Multicast GID FF02:0:0:0:0:0:0:1 is the link-local multicast GID - a router should not route packets with this destination GID outside the local subnet. This GID is used as the destination address within the global router header (GRH) for communicating to a set of QPs participating within the all channel adapters multicast group. ALL CHANNEL ADAPTERS MULTICAST GROUP is used to implement a broadcast service to all channel adapters which are capable of participating in multicast operations.

16) IPv6 defines a set of reserved multicast addresses in RFC 2375 and RFC 2373. IBA, unless explicitly stated otherwise, shall not use these

addresses for IBA multicast operations and defines them as reserved for raw IPv6 usage.

4.1.2 Channel Adapter, Switch, and Router Addressing Rules

C4-4: Channel Adapters, Switches, and Routers shall comply with the addressing rules defined within 4.1.2 Channel Adapter, Switch, and Router Addressing Rules on page 121.

Addressing rules are:

1) A port shall attach to one link.
2) A CA or router port shall support a range of LIDs as defined by a Base LID and an LMC. The LIDs shall be sequentially ordered starting with a base LID plus ($2^{LMC - 1}$) LIDs. The SM may program the LMC on a port to any value between 0 and 7, to allow use of multiple LIDs (1-128) in addressing the port.
3) Switch port 0 shall be assigned a single unicast LID, i.e. LMC = 0.
4) A unicast LID shall map to only one port on a CA or router or to switch port 0.
5) A multicast LID shall map to one or more ports - a port may be the target of zero, one, or more multicast flows.
6) Unicast GIDs shall be assigned to switch port 0 and on a per port basis for CAs and router.
7) A multiport CA (and by definition, a router) may be attached to one or more subnets - a port shall only be attached to one subnet at a time.

4.1.3 Local Identifiers

C4-5: Local Identifiers (LIDs) shall comply with the rules defined within 4.1.3 Local Identifiers on page 121.

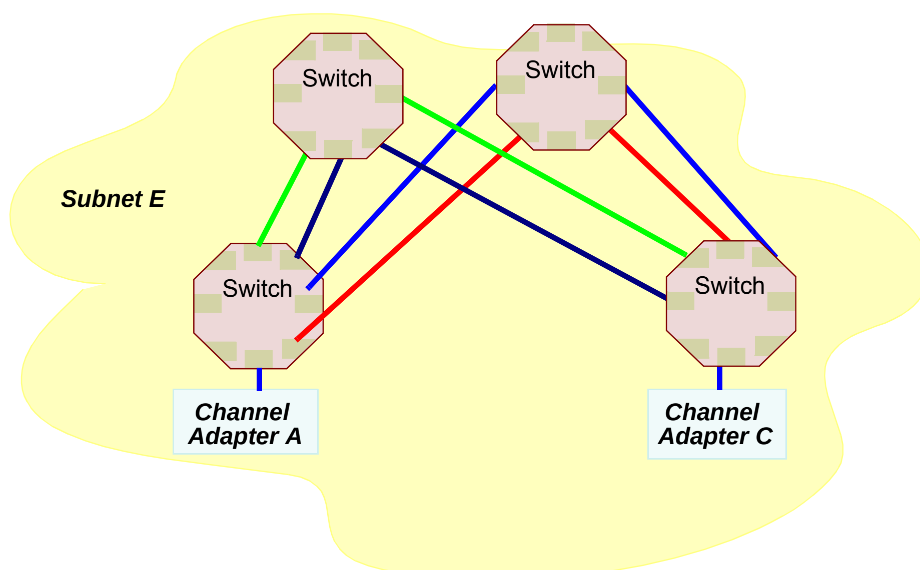
Local identifier (LID): A 16-bit identifier with the following properties:

1) A LID is assigned by the Subnet Manager (SM) and is subnet unique, i.e. it cannot be used to route between subnets.
2) The LID address space is divided into reserved, unicast and multicast address ranges.
3) LIDs are contained within the LRH (Local Route Header).
4) For a CA or router, a source LID (SLID) shall refer to the port that first injected the packet into the subnet. For a switch initiating a packet, the SLID shall be the LID associated with that switch.
5) A SLID shall only be associated with a unicast address.
6) A unicast destination LID (DLID) shall refer to the destination port of a CA or router or to switch port 0. A multicast DLID refers to the set of

destination ports within the subnet participating in a given multicast group.

7) If the destination endnode is not on the same subnet, the DLID shall refer to the router port responsible for forwarding the packet to the next hop to the destination endnode.

8) From any point within a subnet, a given channel adapter or router port may receive packets through multiple physical paths within the subnet. Each physical path may be identified by one or more destination LIDs. To facilitate multipath operation while minimizing channel adapter complexity, each CA and router port and switch port 0 shall be assigned a base LID and a LID Mask Control (LMC) value by the subnet manager. The LMC is a 3-bit field which represents 2^{LMC} paths (maximum of 128 paths). During discovery, the subnet manager may determine the number of paths to a given port and will partition the 16-bit LID space to assign a base LID and up to 2^{LMC} sequential LIDs to each port. Note: The base LID must have LMC least



Four paths between channel adapters A and C. CA A is assigned a Base LID 4, LMC = 2. This translates to CA A being assigned LIDs: {4, 5, 6, 7}. CA C is assigned Base LID 8, LMC = 2. This translates into CA C being assigned LIDs: {8, 9, 10, 11}.

Figure 43 Multipath Identification

significant bits set to 0. For example, if the LMC = 0, the base LID may be any unicast LID. If the LMC = 7, the base LID set the 7 least significant bits to zero.

9) The LID space is defined as follows:

- LID 0x0000 is reserved.
- LID 0xFFFF is defined as a permissive DLID. The permissive DLID indicates that the packet is destined for QP0 on the channel adapter or router port or switch which received it. LMC is not defined for this address.
- The unicast LID range is a flat identifier space defined as 0x0001 to 0xBFFF.
- The multicast LID range is a flat identifier space defined as 0xC000 to 0xFFFE.

Chapter 5: Data Packet Format

This chapter introduces the fields in the data packet. A brief description of each field is given including a definition, field size, and abbreviation. This chapter does not specify the details of each field, but only the general usage and layout of the fields.

In addition to data packets, IBA defines link packets which are used for link-level flow control. The format of these link packets is described in 7.9.4 Flow Control Packet on page 183.

In this specification, the term packet refers to data packets only (i.e. packet and data packet are synonymous). Where reference to link packets is intended, the full term *link packet* will be used.

5.1 Packet Types

Packets are the unit of transfer in IBA. As described in 3.3 Communications Stack on page 69 messages are segmented into packets by the CAs for transmission across the IB fabric.

Packets have the following attributes:

- Indivisible unit of data transfer and routing
- Unit of acknowledgement
- Unit of segmentation and re-assembly for messages
- Unit of link-level flow control

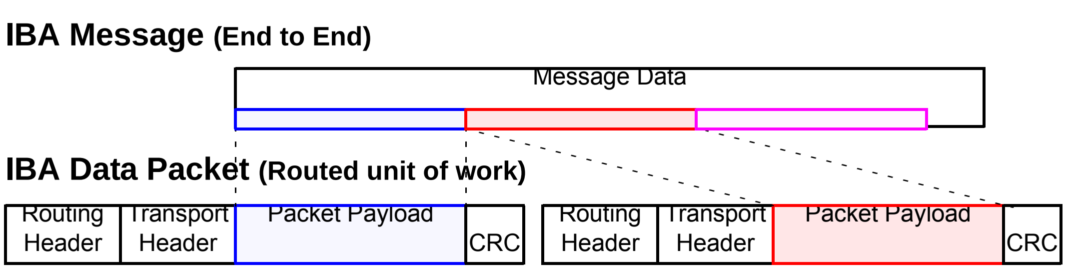


Figure 44 IBA Messages and Packets

There are two general classes of transports used in Packets:

- **IBA Packets** have IBA defined transport headers, are routed on IBA fabrics, and use native IBA transport facilities.
- **Raw Packets** may be routed on IBA fabrics but do not contain IBA transport headers. From the IB point of view, these packets contain only IBA routing headers, payload and CRC. IBA does not define the processing of these packets above the link and network layers. The intent is that these packets can be used to support non-IBA transports over an IB fabric.

5.2 DATA PACKET FORMAT

The overall data packet structure is shown in Figure 45 on page 126. There are two routing headers that precede a transport header(s) and payload:

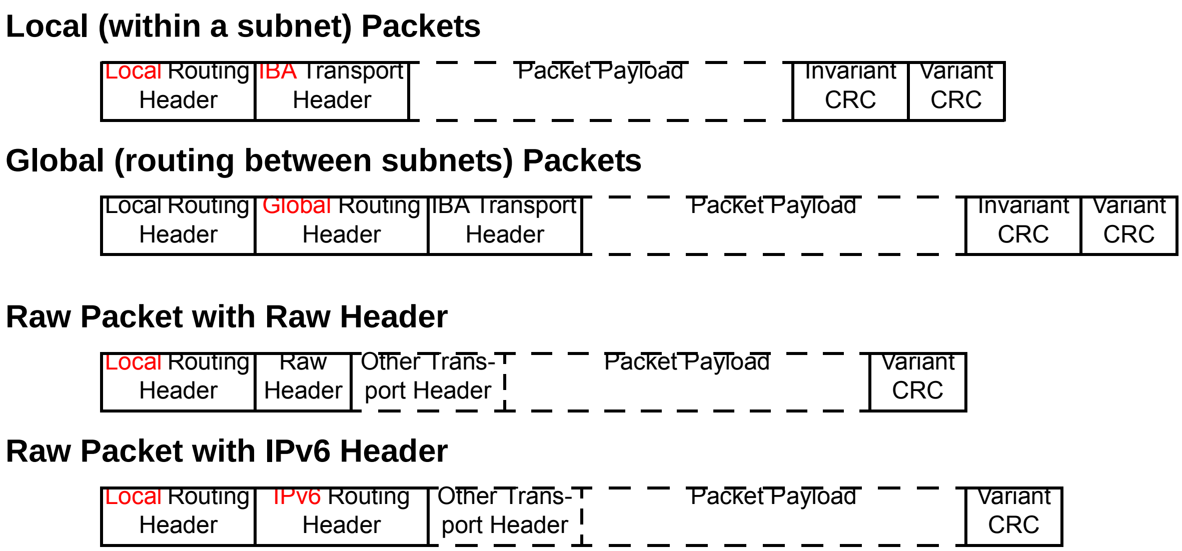
- The local route header is required on all packets
- The global route header is required on all packets that are to be routed to a different subnet, and on all multicast packets regardless of destination.
- A global route header may be placed on any packet except subnet management packets.

C5-1: Packets generated by an InfiniBand device shall conform to the packet structure defined in Figure 45 and to the packet header location and size requirements as defined in figure 46

Each IBA packet ends with an invariant CRC followed by a variant CRC.

Each raw packet ends with a variant CRC.

Figure 45 IBA Packet Overview



The IBA packet structure is shown in Figure 46 on page 127.

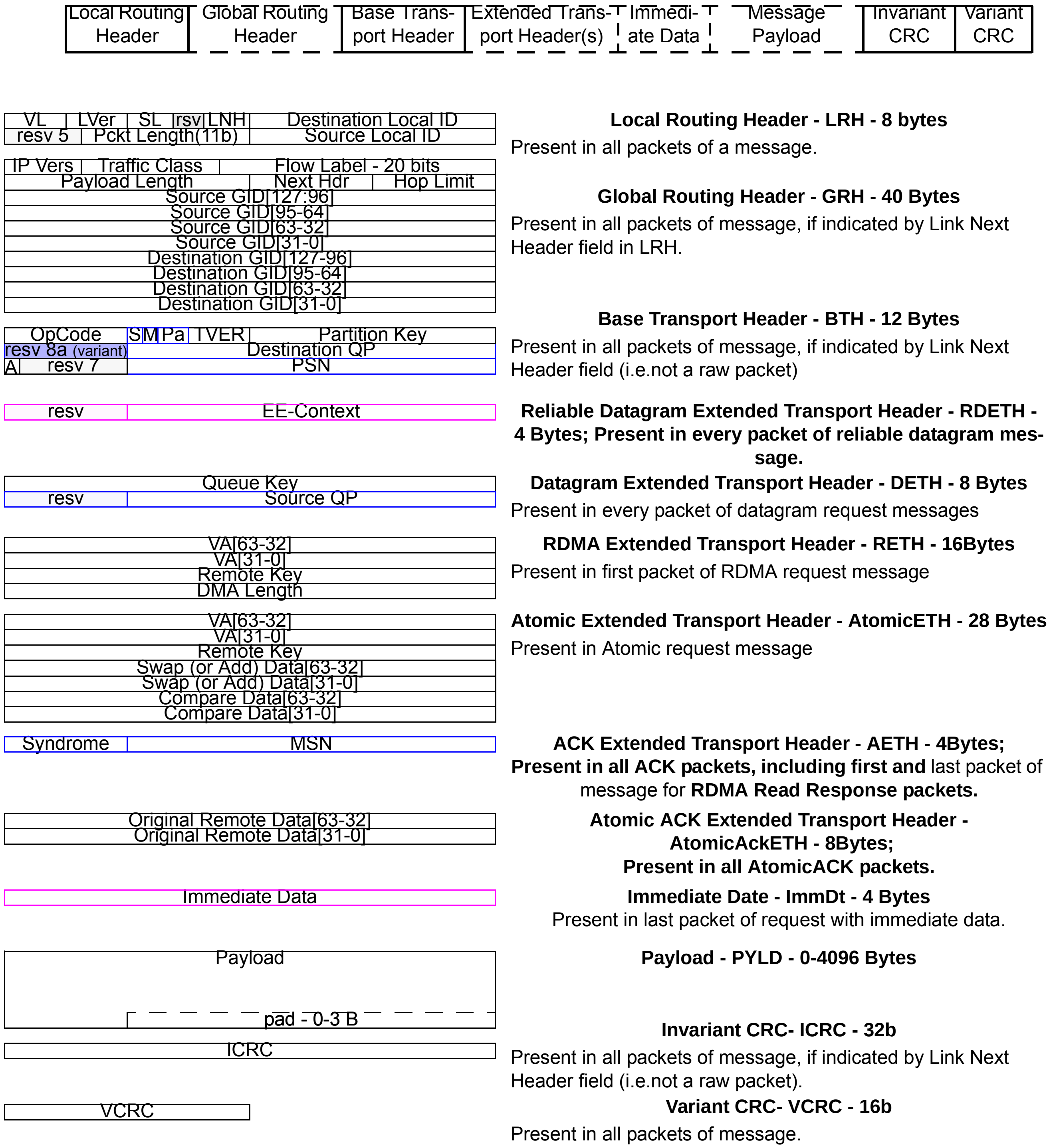


Figure 46 IBA Packet Structure

5.2.1 LOCAL ROUTE HEADER (LRH) - 8 BYTES

C5-2: Packets generated by an InfiniBand device shall conform to the packet header format for the LRH as defined in table 4.

The Local Routing Header (LRH) contains fields used for local routing by switches within a IBA subnet. The following table summarizes the fields in the LRH.:

Table 4 Local Route Header Fields

Field Name	Field Abbre viation	Field Size (in bits)	Description
Virtual Lane	VL	4	This field identifies the virtual lane that the packet is using.
Link Version	LVer	4	This field identifies the Link level protocol of this packet. This version applies to the general packet structure including the LRH fields and the variant CRC
Service Level	SL	4	This field indicates what service level the packet is requesting within the subnet.
Reserved		2	Transmitted as 0, ignored on receive.
Link Next Header	LNH	2	This field identifies the headers that follow the LRH.
Destination Local ID	DLID	16	This field identifies the destination port and path (data sink) on the local subnet.
Reserved		5	Transmitted as 0, ignored on receive.
Packet Length	PktLen	11	This field identifies the size of the Packet in four-byte words. This field includes the first byte of LRH to the last byte before the variant CRC. See 7.7.8 Packet Length (PktLen) - 11 bits on page 167 for details on max and min values of PktLen
Source Local ID	SLID	16	This field identifies the source port (injection point) on the local subnet.

The LRH fields are fully defined in 7.7 Local Route Header on page 165.

5.2.2 GLOBAL ROUTE HEADER (GRH) - 40 BYTES

C5-3: Packets generated by InfiniBand devices shall conform to the packet header format for the GRH as defined in table 5.

Global Route Header (GRH) contains fields for routing the packet between subnets. The presence of the GRH is indicated by the Link Next Header (LNH) field in the LRH. The layout of the GRH is the same as the IPv6 Header defined in RFC 2460. Note, however, that IBA does not define a relationship between a device GID and IPv6 address (I.e. there is no defined mapping between GID and IPv6 address for any IB device or port).

The following table summarizes the fields in the GRH.

Table 5 Global Route Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
IP Version	IPVer	4	This field indicates version of the GRH
Traffic Class	TClass	8	This field is used by IBA to communicate global service level.
Flow Label	Flow-Label	20	This field identifies sequences of packets requiring special handling.
Payload length	PayLen	16	For an IBA packet this field specifies the number of bytes starting from the first byte after the GRH, up to and including the last byte of the ICRC. For a raw IPv6 datagram this field specifies the number of bytes starting from the first byte after the GRH, up to but not including either the VCRC or any padding, to achieve a multiple of 4 byte packet length. For raw IPv6 datagrams padding is determined from the lower 2 bits of this GRH:PayLen field. Note: GRH:PayLen is different from LRH:PkyLen.
Next Header	NxtHdr	8	This field identifies the header following the GRH. This field is included for compatibility with IPV6 headers. It should indicate IBA transport.
Hop Limit	HopLmt	8	This field sets a strict bound on the number of hops between subnets a packet can make before being discarded. This is enforced only by routers.
Source GID	SGID	128	This field identifies the Global Identifier (GID) for the port which injected the packet into the network.
Destination GID	DGID	128	This field identifies the GID for the port which will consume the packet from the network.

5.2.3 BASE TRANSPORT HEADER (BTH) - 12 BYTES

C5-4: Packets generated by an Infiniband device shall conform to the packet header format for the BTH as defined in table 6.

Base Transport Header (BTH) contains the fields for IBA transports. The presence of BTH is indicated by the Next Header field of the last previous header (i.e either LRH:LNH or GRH:NextHdr depending on which was the last previous header). The following table summarizes the fields in the BTH.:

Table 6 Base Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Opcode	OpCode	8	This field indicates the IBA packet type. The OpCode also specifies which extension headers follow the Base Transport Header
Solicited Event	SE	1	This bit indicates that an event should be generated by the responder.
MigReq	M	1	This bit is used to communicate migration state.
Pad Count	PadCnt	2	This field indicates how many extra bytes are added to the payload to align to a 4 byte boundary.
Transport Header Version	TVer	4	This field indicates the version of the IBA Transport Headers.
Partition Key	P_KEY	16	This field indicates which logical Partition is associated with this packet (see 10.9 Partitioning on page 454)
Reserved (variant)		8	Transmitted as 0, ignored on receive. This field is not included in the invariant CRC. see 7.8 CRCs on page 168 for details.
Destination QP	DestQP	24	This field indicates the Work Queue Pair Number (a.k.a. QP) at the destination
Acknowledge Request	A	1	This bit is used to indicate that an acknowledge (for this packet) should be scheduled by the responder.
Reserved		7	Transmitted as 0, ignored on receive. This field is included in the invariant CRC.
Packet Sequence Number	PSN	24	This field is used to detect a missing or duplicate Packet. See 9.7.1 Packet Sequence Numbers (PSN) on page 248 for a detailed description of PSN.

The detailed definition of the Base Transport Header fields are defined in Section 9.2 on page 206.

5.2.4 Reliable Datagram Extended Transport Header (RDETH) - 4 Bytes

o5-1: Packets generated by an Infiniband device that supports reliable datagrams shall conform to the packet header format for the RDETH header as defined in table 7.

Reliable Datagram Extended Transport Header (RDETH) contains the additional transport fields for reliable datagram service. The RDETH is only in Reliable Datagram packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the RDETH.:

Table 7 Reliable Datagram Extended Transport Header Fields

Field Name	Field Abbre viation	Field Size (in bits)	Description
Reserved		8	Transmitted as 0, ignored on receive.
EE-Context	EECnxt	24	This field indicates which End-to-End Context should be used for this Reliable Datagram packet

The detailed definition of the Reliable Datagram Extended Transport Header is in Section 9.3.1 Reliable Datagram Extended Transport Header (RDETH) - 4 Bytes on page 210.

5.2.5 DATAGRAM EXTENDED TRANSPORT HEADER (DETH) - 8 BYTES

C5-5: Packets generated by an Infiniband device shall conform to the packet header format for the DETH as defined in table 8.

Datagram Extended Transport Header (DETH) contains the additional transport fields for datagram service. The DETH is only in datagram packets if indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the DETH.:

Table 8 Datagram Extended Transport Header Fields

Field Name	Field Abbre viation	Field Size (in bits)	Description
Queue Key	Q_Key	32	This field is required to authorize access to the receive queue.
Reserved		8	Transmitted as 0, ignored on receive.
Source QP	SrcQP	24	This field indicates the Work Queue Pair Number (a.k.a. QP) at the source.

The detailed definition of the Datagram Extended Transport Header is in Section 9.3.2 Datagram Extended Transport Header (DETH) - 8 Bytes on page 211.

5.2.6 RDMA EXTENDED TRANSPORT HEADER (RETH) - 16 BYTES

o5-2: Packets generated by an Infiniband device that supports RDMA operations shall conform to the packet header format for the RETH as defined in table 9.

RDMA Extended Transport Header (RETH) contains the additional transport fields for RDMA operations. The RETH is present in only the first (or only) packet of an RDMA Request as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the RETH.:

Table 9 RDMA Extended Transport Header Fields

Field Name	Field Abbre viation	Field Size (in bits)	Description
Virtual Address	VA	64	This field is the Virtual Address of the RDMA operation.
Remote Key	R_Key	32	This field is the Remote Key that authorizes access for the RDMA operation.
DMA Length	DMALen	32	This field indicates the length (in Bytes) of the DMA operation.

The detailed definition of the RDMA Extended Transport Header is in 9.3.3 RDMA Extended Transport Header (RETH) - 16 Bytes on page 212.

5.2.7 ATOMIC EXTENDED TRANSPORT HEADER (ATOMICETH) - 28 BYTES

o5-3: Packets generated by an Infiniband device that supports atomic operations shall conform to the packet header format for the AtomicETH header as defined in Table 10.

Atomic Extended Transport Header (AtomicETH) contains the additional transport fields for Atomic packets. The AtomicETH is only in Atomic packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the AtomicETH.:

Table 10 Atomic Extended Transport Header Fields

Field Name	Field Abbre viation	Field Size (in bits)	Description
Virtual Address	VA	64	This field is the remote virtual address.
Remote Key	R_Key	32	This field is the Remote Key that authorizes access to the remote virtual address.
Swap (or Add) Data	SwapDt	64	This field is an operand in atomic operations.
Compare Data	CmpDt	64	This field is an operand in CmpSwap atomic operation.

The detailed definition of the Atomic Extended Transport Header is in Section 9.3.4 ATOMIC Extended Transport Header (AtomicETH) - 28 Bytes on page 213).

5.2.8 ACK EXTENDED TRANSPORT HEADER (AETH) - 4 BYTES

C5-6: Packets generated by an Infiniband device shall conform to the packet header format for the AETH as defined in table 11.

ACK Extended Transport Header (AETH) contains the additional transport fields for ACK packets. The AETH is only in Acknowledge, RDMA READ Response First, RDMA READ Response Last, and RDMA READ Response Only packets as indicated by the Base Transport Header Opcode field. The following table summarizes the fields in the AETH.

Table 11 ACK Extended Transport Header Fields

Field Name	Field Abbreviation	Field Size (in bits)	Description
Syndrome	Syndrome	8	This field indicates if this is an ACK or NAK packet plus additional information about the ACK or NAK.
Message Sequence Number	MSN	24	This field indicates the sequence number of the last message completed at the responder.

The detailed definition of the ACK Extended Transport Header is in Section 9.3.5 on page 214.

5.2.9 ATOMIC ACK EXTENDED TRANSPORT HEADER (ATOMICACKETH) - 8 BYTES

o5-4: Packets generated by an Infiniband device that supports atomic operations shall conform to the packet header format for the AtomicAckETH as defined in table 12.

Atomic ACK Extended Transport Header (AtomicAckETH) contains the additional transport fields for AtomicACK packets. The AtomicAckETH is only in Atomic Acknowledge packets as indicated by the Base Transport

Header Opcode field. The following table summarizes the fields in the AtomicAckETH.:.

Table 12 Atomic ACK Extended Transport Header Fields

Field Name	Field Abbre viation	Field Size (in bits)	Description
Original Remote Data	Orig-RemDt	64	This field is the return operand in atomic operations and contains the data in the remote memory location before the atomic operation.

The detailed definition of the Atomic ACK Extended Transport Header is in Section 9.3.5.3 on page 214.

5.2.10 IMMEDIATE DATA EXTENDED TRANSPORT HEADER (IMMDT) - 4 BYTES

Immediate DataExtended Transport Header (ImmDt) contains the additional data that is placed in the receive Completion Queue Element (CQE). The ImmDt is only in Send or RDMA-Write packets with Immediate Data if indicated by the Base Transport Header Opcode.

The detailed definition of the Immediate Data Extended Transport Header is in Section 9.3.6 on page 215.

Note, the terms *Immediate Data Extended Transport Header* and *Immediate Data* are used synonymous in this specification.

5.2.11 PAYLOAD

Payload (PYLD) contains the application data being transferred end to end. Payload is not present in RDMA Read Requests, Acknowledge, CmpSwp, FetchAdd, and Atomic Acknowledge packets. It is optionally present in the other packet op-codes.

C5-7: The length of the Payload shall be 0 or more bytes up to the full path MTU.

C5-8: All packets of an IBA message that contain a payload shall fill the payload to the full path MTU except the last (or only) packet of the message.

C5-9: In a packet using InfiniBand transport, a Pad field of 0-3 bytes shall be included in the packet and used to align the Payload to a multiple of 4 bytes (i.e. the size of the Payload plus the Pad field is always a multiple of four bytes). The actual size of the Pad field used in a given packet shall be indicated in the Base Transport Header PadCnt field of the packet.

5.2.12 INVARIANT CRC

Invariant CRC (ICRC) covers the fields that do not change in packet from source to destination. ICRC is only in IBA packets, and is not present in Raw Packets. Which fields are covered in the ICRC is dependent on the presence of the GRH.

The detailed definition of the Invariant CRC is in Section 7.8.1 on page 168.

5.2.13 VARIANT CRC

Variant CRC (VCRC) covers the fields that can change from link to link. The VCRC is in all packets, both IBA and Raw Packets. The VCRC can be regenerated in the fabric.

The detailed definition of the Variant CRC is in Section 7.8.2 on page 170.

5.3 RAW PACKET FORMAT

A Raw Packet is a packet that does not use IBA transport. Raw packets are not a required feature of InfiniBand devices, but if they are supported, the raw packet shall be formatted as specified in this section.

o5-5: If a Raw packet contains an IPv6 Routing Header, the packet structure shall be: LRH, IPv6, Payload (including any transport headers), and VCRC. If a Raw packet does not contain a IPv6 Routing Header, then the structure shall be: LRH, RWH, Payload, and VCRC.

o5-6: The RWH is a 32 bit “Raw Header” that shall contain the EtherType of the payload. EtherType indicates the protocol of the raw packet and shall conform to the definition in the IEEE Type Field Registrar. (See standards IEEE 802.3, 1998 Clause 3.2.6 Length/Type Field specifications and IEEE 802.1H-1995 for use of the Type Field.)

This format of “Raw” packets is shown in Figure 45 on page 126.

o5-7: The length of a raw packet (from after the RWH to before the variant CRC) must be a multiple of 4 bytes.

o5-8: The format of the Raw Header shall be as is shown in Figure 47.

Figure 47 Raw Header (RWH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Reserved (Send as 0, ignore on receive)		EtherType	

5.4 PACKET EXAMPLES

Some examples of IBA packets are shown in Figure 48.

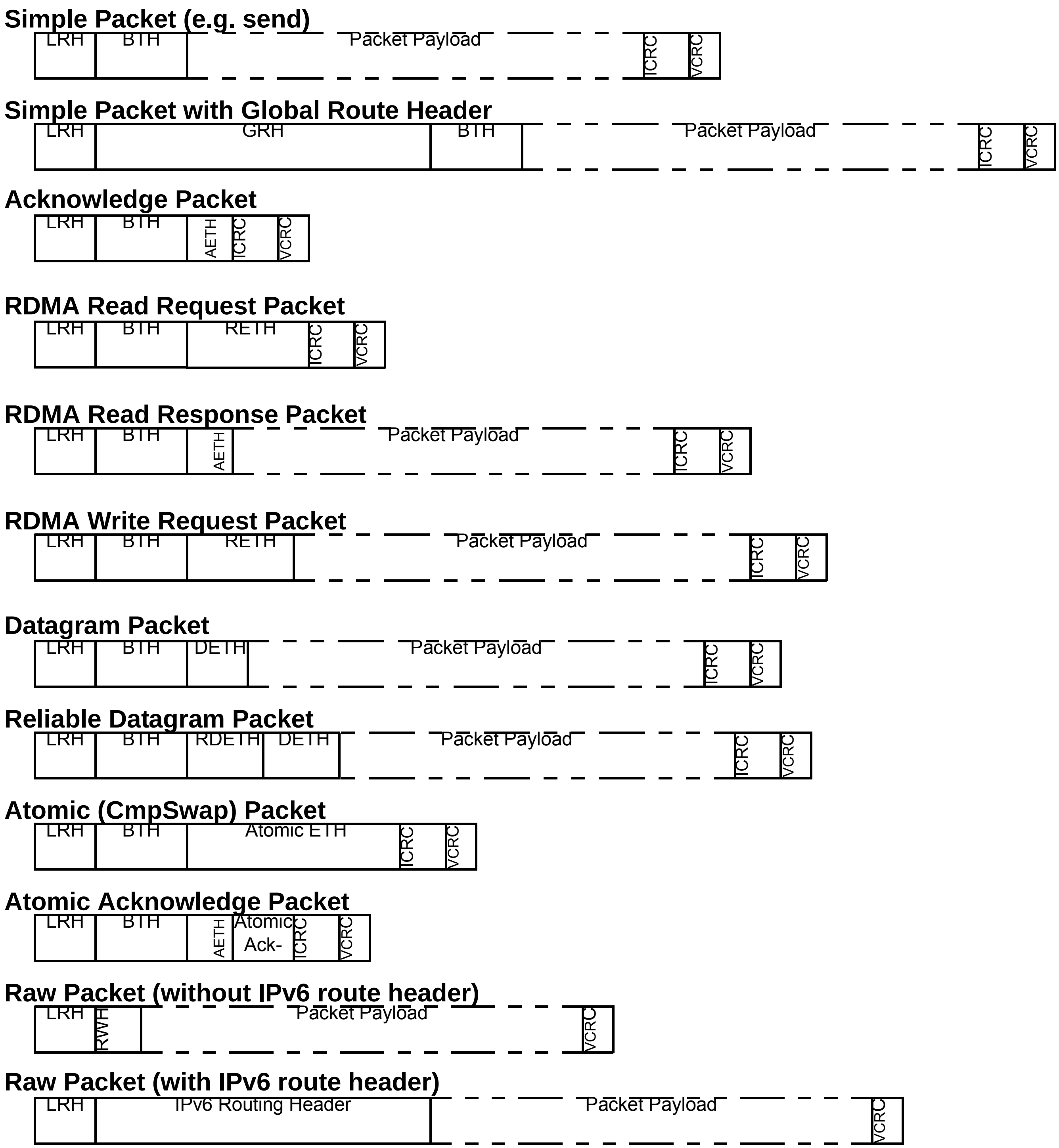


Figure 48 IBA Packet Examples

CHAPTER 6: PHYSICAL LAYER INTERFACE

6.1 OVERVIEW

This chapter describes services provided by the physical layer to the link layer and the logical interface between these layers. The physical layer also has an interface to management which is not covered in this chapter.

The description of the physical layer is provided in Volume 2, the electro-mechanical specification

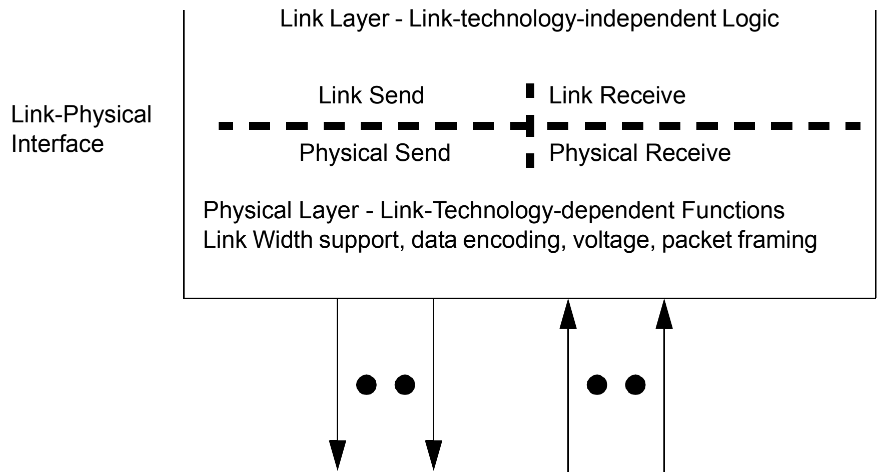


Figure 49 Physical Functions and Physical/Link Interface

6.2 SERVICES PROVIDED BY THE PHYSICAL LAYER.

The physical layer is responsible for:

- establishing a physical link when possible,
- informing the link layer whether the physical link is up or down,
- monitoring the status of the physical link, and
- when the physical link is up:
  - delivering received control and data bytes to the link layer, and
  - transmitting control and data bytes from the link layer.

See volume 2 for physical layer specifications.

6.3 INTERFACE BETWEEN PHYSICAL AND LINK LAYERS.

This chapter does not intend to describe an actual interface within a chip - it describes the functionality of the interface between the link-technology-dependent physical send and receive functions, and the link-technology-independent link logical function.

This interface is designed to keep the link and higher layer interface independent of physical layer implementation. The physical layer deals with all details that are dependent on the characteristics of operation over a particular physical layer such as line code.

The purpose of describing a logical interface and the related state machines is to partition functions to describe external behavior of IBA devices as simply and clearly as possible. Such descriptions are not intended to imply details of the internal implementation of devices. For instance, the interface described here does not imply the width of the internal link path which will be implementation dependent.

6.3.1 INTERFACE BETWEEN PHYSICAL RECEIVE AND LINK RECEIVE.

The following messages are sent between the physical receive function and the link logic.

6.3.1.1 PHY_LINK - PHYSICAL LINK STATUS

This message conveys the status of the physical link from the physical receive function to the link logic. This message is sent when physical link status changes and can take the following values:

down	the physical link is not operational. Sent when the link is in any non-operational status including no receive signal or retraining in progress
up	the physical link is trained and operational

These values report the status of the physical link as needed by the link logic. Any finer grain information needed by management (e.g. no_signal or retraining) will be obtained by management from the physical layer rather than passed through the link layer.

6.3.1.2 L_INIT_TRAIN - LINK INITIATE RETRAINING

This message is a request for retraining of the physical link. It is sent from the link logic to the physical receive function when the link logic has detected a need to retrain the link. See Section 7.12.2, "Error Recovery Procedures," on page 194 for usage of this message.

6.3.1.3 RCV_STREAM - Receive Stream

This message conveys the control and data stream decoded by the receiver from the physical receive function to the link logic. This message is sent once for each data byte and once for each control signal received. The idle signaling of the physical link is treated as one control signal. This message can take the following values:

data	data and link packet contents
error	code violation
SDP	start data packet delimiter
SLP	start link packet delimiter
EGP	end good packet delimiter
EBP	end bad packet delimiter
idle	idle

6.3.2 Interface between physical Transmit and link Transmit.

The following messages are sent between the physical transmit function and the link logic.

6.3.2.1 XMIT_STREAM - Transmit Stream

This message conveys the control and data stream from the link logic to the physical layer. This message is sent once for each data byte and once for each control signal to be sent. The idle message causes the physical send function to send idles until a new message is received. This message can take the following values:

data	data and link packet contents
SDP	start data packet delimiter
SLP	start link packet delimiter
EGP	end good packet delimiter
EBP	end bad packet delimiter
idle	idle

6.3.2.2 XMIT_READY - Physical Transmitter Ready

This message is sent from the physical transmit function to the link transmitter to indicate whether the physical transmit function is ready to start transmitting a new packet. This provides physical layer dependent pacing back to the link layer since many physical layers have constraints that prevent sending continuous packet traffic. This message can take the following values:

rdy	ready for packet initiation
wait	hold off packet initiation

CHAPTER 7: LINK LAYER

7.1 OVERVIEW

This chapter describes the behavior of the link and specifies the link level operations for devices attached to an IBA network. The link layer handles the sending and receiving of data across the links at the packet level. Services provided by the link layer include addressing, buffering, flow control, error detection and switching.

State machines are used in this specification to define the logical operation of the link layer as externally visible. They are not intended to define internal details of implementation. For instance, the packet receiver state machine operates on data received from the link layer as a stream of bytes though it is expected that many implementations of the link layer will process multiple bytes of the data stream in parallel.

7.1.1 STATE MACHINE CONVENTIONS

State machines are described to provide a clear description of the external behavior of the devices. Their description is not intended to imply the internal implementation of IBA devices. Actual implementations must take into account other considerations such as efficiency and suitability to the implementation technology.

The state machines in this chapter use the following conventions:

- Each state is represented by a box.
- The top section of the box contains the state name.
- The bottom section of the box contains the actions which occur in the state.
- Transition arrows indicate state transitions which will be made when the expression next to the arrow is satisfied.
- A transition arrow which does not originate in a state indicates a global transition. Such a transition will occur regardless of the current state. For instance, in Figure 50 on page 144, there is a global transition into the LinkDown state.
- If no exit condition for a state is satisfied, the machine remains in the current state.
- "Or" is represented by "+".
- "And" is represented by "*".

- The state diagrams represent the primary specification for the functions they depict. When a conflict exists between a state diagram and descriptive text, the state diagram takes precedence.

7.2 LINK STATES

C7-1: A port shall control its state and overall operation as specified in Figure 50 Link State Machine on page 144 and Section 7.2.7, "State Machine Terms," on page 143.

The states LinkInitialize and LinkArm are used by subnet management to configure devices on the subnet. Refer to 14.3.6 Port State Change on page 684 for additional information on how these states are used.

The link state machine is depicted in Figure 50. The following is a description of the states of this state machine.

7.2.1 LINKDOWN STATE

In the LinkDown state, the physical link is not up (that is, the physical layer is sending phy_link=down to the link layer) and the link layer is idle. In this state the link layer discards all packets presented to it for transmission.

7.2.2 LINKINITIALIZE STATE

In the LinkInitialize state, the physical link is up (that is, the physical layer is sending phy_link=up to the link layer) and the link layer can only receive and transmit subnet management packets (SMPs) and flow control link packets. While in this state, the link layer discards all other packets received or presented to it for transmission.

7.2.3 LINKARM STATE

In the LinkArm state, the physical link is up and the link layer can receive and transmit SMPs and flow control link packets. Additionally, the link layer can receive all other packets but discards all non-SMP data packets presented to it for transmission.

A switch port which is moved from LinkArm to LinkActive by a packet may also be the output port for that packet. The port will not be armed until the CRC has been checked for the packet. One data packet should be able to pass through the network causing the armed ports in its path to transition to active. Therefore, it is important that such a packet not be dropped due when if it is forwarded to the port before the port has transitioned to active.

C7-1.a1: A switch shall ensure that a packet which causes its output port to transition from Armed to Active is not dropped by the port while in the Armed state. A switch port may enable transmission of data packets while in the Armed state.

7.2.4 LinkActive State

In the LinkActive state, the physical link is up and the link layer can transmit and receive all packet types.

7.2.5 LinkActDefer State

The LinkActDefer state is entered from the LinkActive state when the physical layer indicates a failure in the link. If the error persists, the LinkDownTimeout expires and the port state transitions to LinkDown state. If the physical layer recovers prior to LinkDownTimeout expiration, the port state machine returns to the LinkActive state. While in the LinkActDefer state, the link layer will not transmit or receive packets. It may process packets already received as it would in the corresponding states. It will drop packets presented to it for transmission.

The purpose of this state is to allow for retraining of the physical link without requiring reinitialization of the link and higher layers.

7.2.6 Management State Change Commands

Management can send commands to attempt to alter the link state by sending a set request to the link port state in PortInfo. Only values of Down, Arm and Active are valid for such set requests. Commands to change state to Arm or Active are only valid when they appear as an exit term for the current state.

C7-2: Any management state change command with a value other than Down, Arm, or Active shall not result in a state change.

C7-3: A management state change command which is not valid in the current state shall not result in a state change.

For instance, Active is only valid when the current state is LinkArm. If the command is not valid for the current state, it will not cause a state change.

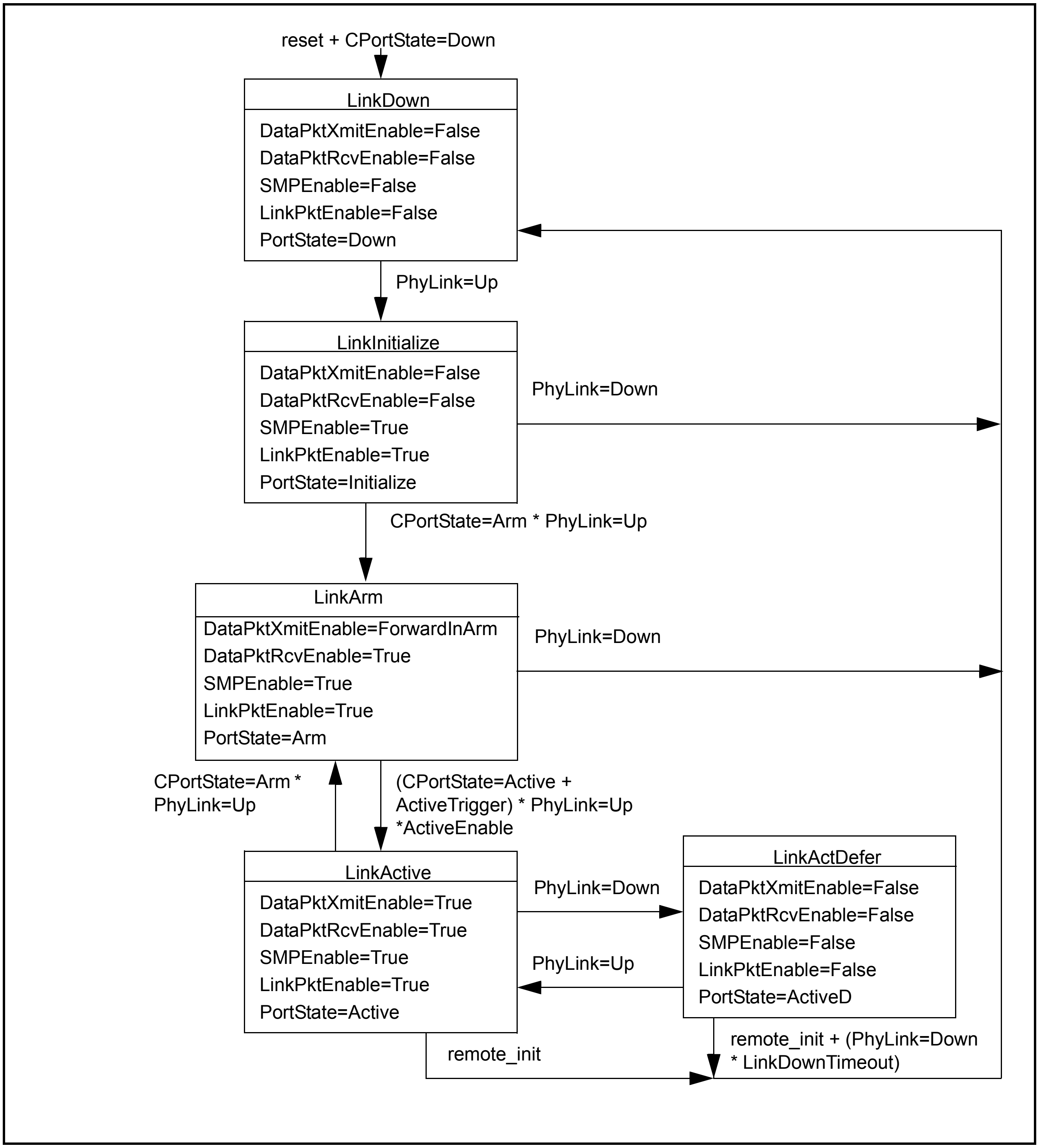
7.2.7 State Machine Terms

Reset - An internal signal to reset the interface.

Remote_init - a link packet with the flow control initialize Op code (see 7.9.4 Flow Control Packet on page 183) has been received and has passed the checks of the link packet check state machine.

Active_enable - a flag to prevent a premature transition from armed to active. It is set to false when the LinkInitialize state is exited. It is set to true when a link packet with the normal flow control Op code has been received and has passed the checks of the link packet check state machine while in the LinkArm state.

Figure 50 Link State Machine



PhyLink - the physical link status, phy_link, from the physical layer (refer to 6.3.1.1 Phy_link - Physical Link Status on page 138). Valid values are Up and Down.

PortState - the value of the PortState component of the PortInfo attribute. (Refer to 14.2.5.6 PortInfo on page 665.) Valid values are “Down”, “initialize”, “Arm”, “Active”, and “ActiveD”.

CPortState - a value that indicates commands from management to change the port state. Valid values are “Down”, “Arm”, and “Active”. Note that when phy_link=up and CPortState=down, the state machine will transition to the LinkDown state which will reset other link state machines. Since phy_link=up, this will be followed by a transition to the LinkInitialize state. Thus a command to change link port state to down provides a way to re-initialize the link layer. To disable a port requires a command to the physical layer port state machine. The value of CPortState shall only persist while in the state where it was received. If it satisfies a transition term from that state, it shall cause the transition. If it does not, it shall cause no transitions. Any state transition clears CPortState.

DataPktXmitEnable - a Boolean that indicates the link layer’s action with respect to transmission of non-SMP data packets. When True, transmission of non-SMP data packets is enabled. When False, non-SMP data packets submitted to link layer for transmission are discarded.

DataPktRcvEnable - a Boolean that indicates the link layer’s action with respect to reception of non-SMP data packets from the physical layer. When True, reception of non-SMP data packets is enabled. When False, non-SMP data packets received from the physical layer are discarded.

SMPEnable - a Boolean that indicates the link layer’s action with respect to transmission and reception of subnet management packets (SMPs). When True, transmission and reception of SMPs are enabled. When False, SMPs submitted to link layer for transmission or reception are discarded.

LinkPktEnable - a Boolean that indicates the link layer’s action with respect to transmission and reception of link packets. When True, transmission and reception of link packets are enabled. When False, link packets are not generated by the link layer and any link packets received are discarded.

ForwardInArm - a Boolean constant that indicates whether transmission of data packets is enabled during the Arm state. For a CA, this shall equal False. A switch may optionally use False or True.

AcitveTrigger - a device dependent trigger that initiates the transition from LinkArm to LinkActive. For routers and channel adapters, ActiveTrigger occurs upon reception of a non-VL15 packet which passes the VCRC check on the port. For switches, ActiveTrigger occurs upon reception of a non-VL15 packet which passes the VCRC check on any port of the switch.

LinkDownTimeout - a timeout that indicates that the physical link has been down (PhyLink = down) for a period of time that causes the port state machine to transition to the LinkDown state. LinkDownTimeout occurs when the port state machine has continuously been in the LinkActDefer state for 10ms +3% / -51%.

7.3 PACKET RECEIVER STATES

C7-4: Whenever the physical link is up, the packet receiver shall process the received stream from the physical layer as defined in Figure 51 Packet Receiver State Machine on page 147.

The packet receiver's primary input is the rcv_stream (refer to 6.3.1.3 rcv_stream - Receive Stream on page 139).

The packet receiver monitors the received stream from the physical layer, rcv_stream, and passes any packets received with proper delimiters and no code violations to the link packet check or the data packet check as appropriate. Each byte of the rcv_stream is tested once by the state machine and causes at most one state transition. For example, when an SLP causes a transition from RcvDataPacket to BadPacket, that SLP does not cause a further transition from BadPacket to RcvLinkPacket.

While this logical state machine represents sending the whole packet to the packet checker once the end delimiter is received, implementations are allowed to begin processing the packet before that has occurred. Switches and routers may begin to forward a data packet while in the RcvDataPacket state if the packet passes all checks of the Data Packet Check state machine which require discard of the packet on failure. The required checks are all based on fields within the LRH. If further processing of the packet results in a transition to the MarkedBadPacket or BadPacket states and the switch or router has begun forwarding the packet, the switch or router shall corrupt the packet.

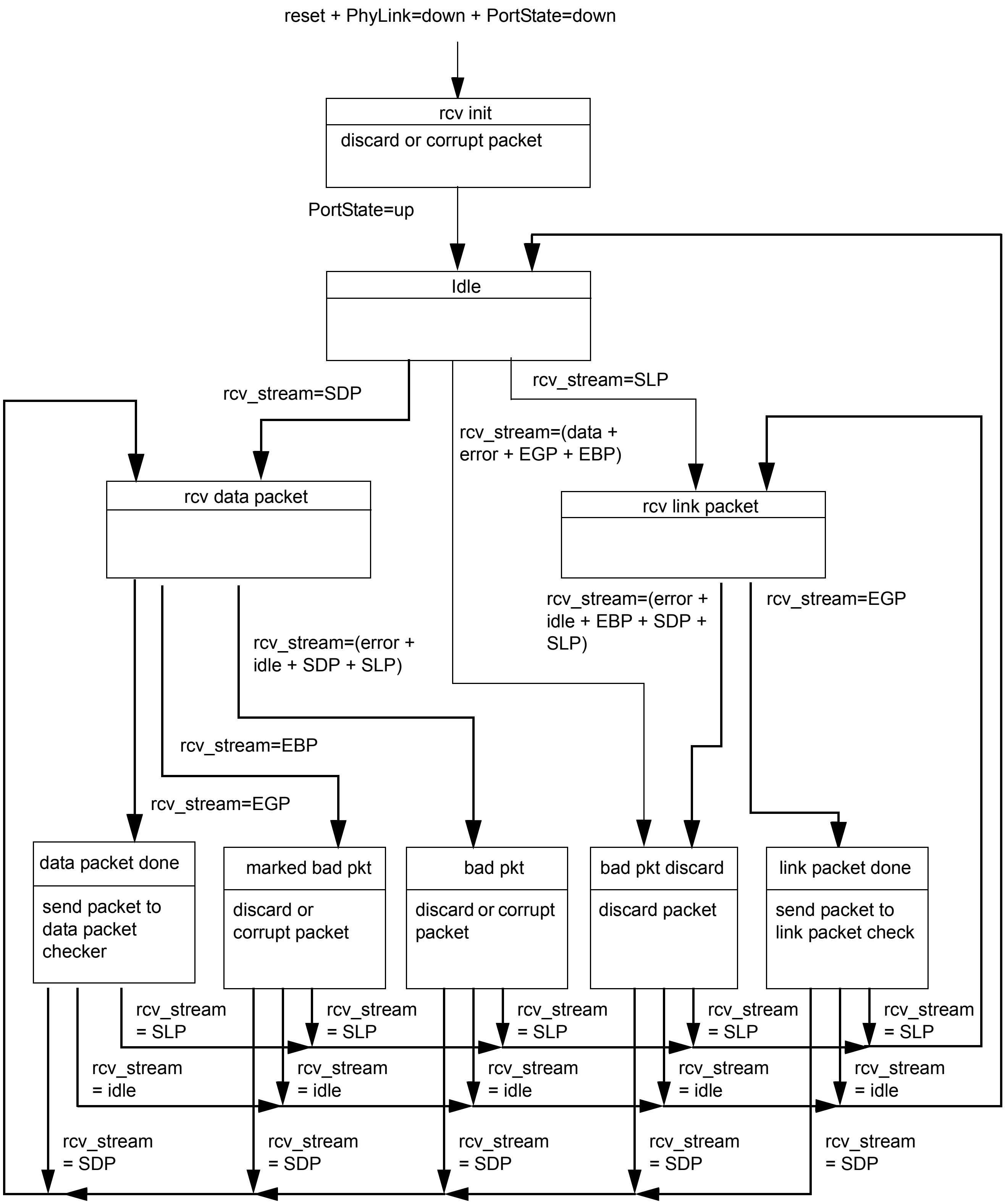
C7-5: To corrupt a packet, a switch or router shall place the 1's complement of the VCRC calculated for the transmitted packet in the VCRC field and shall terminate the packet with the EBP delimiter.

o7-1: When corrupting a packet, the switch or router may truncate the packet rather than sending all the received bytes.

C7-6: If a switch or router is forwarding a corrupted packet which is longer than indicated by the packet length field of the LRH, then it shall truncate the packet to less than or equal to the packet length field value.

C7-7: A CA shall not deliver a received packet to its client unless it has passed all the checks of the packet receiver and data packet check state

Figure 51 Packet Receiver State Machine



machines.Therefore, when the action in the state is “discard or corrupt,” a CA shall discard the packet.

Packets without proper start delimiters cause entry to the bad packet discard state and are discarded. Packets received with one or more bytes of rcv_stream=error or without proper end delimiters cause entry to the bad packet state and are discarded by CAs and discarded or corrupted by switches. The errors which cause entry to the bad packet discard and bad packet states indicate an error occurring on the local link. Packets received with no bytes of rcv_stream=error, a data packet start delimiter (SDP), and a bad packet end delimiter (EBP) indicate a packet forwarded by a switch that experienced an error that was not on the local link. These packets cause entry to the marked bad pkt state. Since link packets are not forwarded by switches and routers, they should never have a bad packet end delimiter. A packet with a start delimiter of SDP and an end delimiter of EBP is considered a local link error and causes entry to the bad packet state.

7.4 Data Packet Check

The data packet check machine in a CA verifies a data packet before passing it to the network layer. The data packet check machine in a switch or router port verifies a received data packet.

C7-8: Data packets shall be checked as specified by Figure 52 Data Packet Check machine on page 149 and Section 7.4, “Data Packet Check,” on page 148. The order of checks within this state machine indicates the precedence of the errors for reporting and not necessarily the order in which the errors are detected.

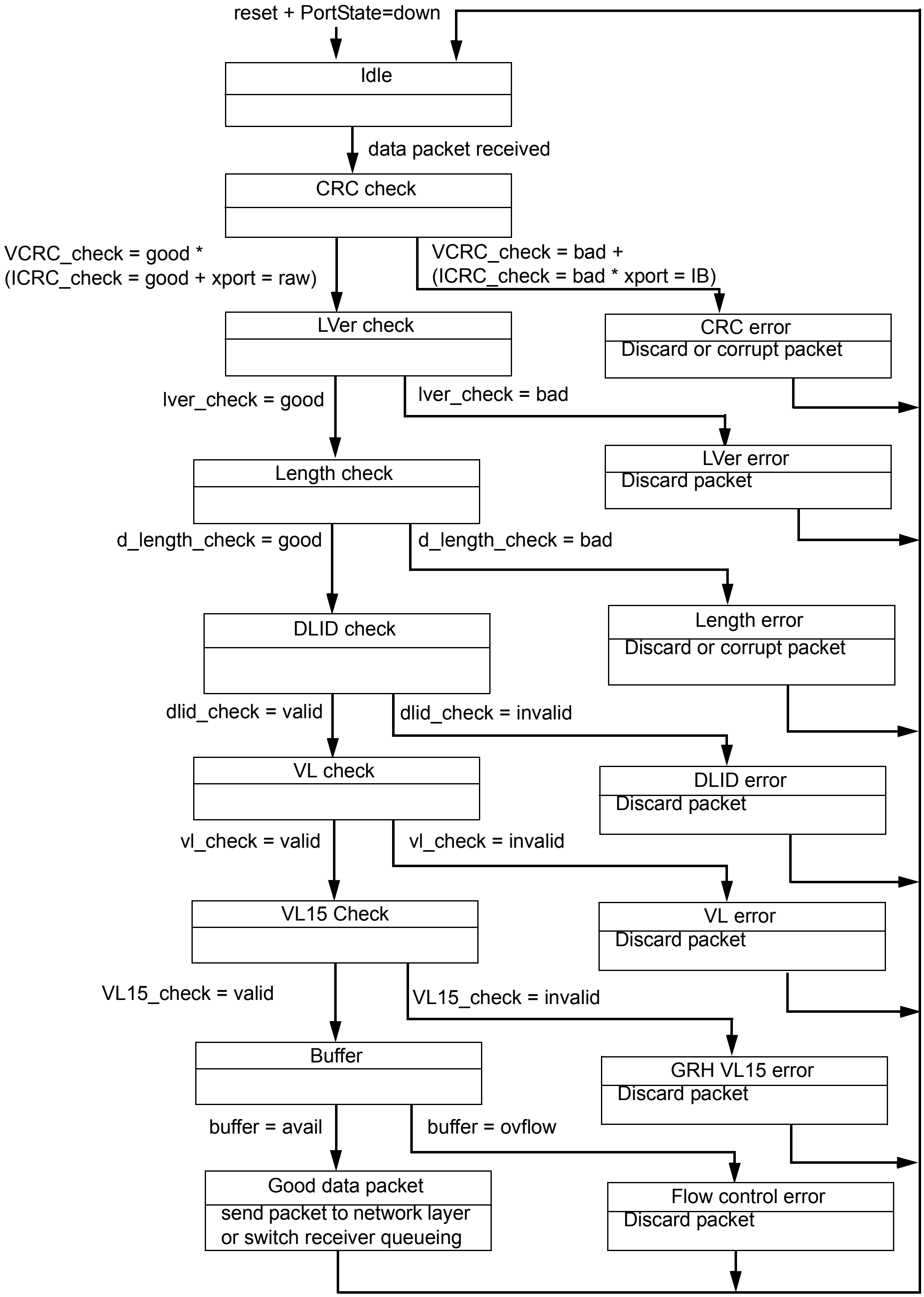
For instance, most implementations would detect an invalid VL shortly after the packet starts and a CRC error cannot be detected until the end of the packet. However, CRC error is checked first in the state machine because if both of these errors occur, the CRC error indicates that the packet was damaged and that error should be reported rather than the VL error.

C7-9: A switch or router shall perform the same checks as a CA on packets for which the switch or router is the destination such as management packets addressed to the switch or router.

The data packet check machine in a CA passes packets to the receiver queueing. See Section 18.2.5.2 Receiver Queuing on page 860.

C7-10: If a packet fails any test that terminates in a state of the Data Packet Check State Machine with the action “discard,” switches, routers, and CAs shall discard the packet.

Figure 52 Data Packet Check machine



C7-11: For packets that only fail tests terminating in states of the Data Packet Check State Machine that specify the action of "corrupt or discard," a CA shall discard the packet and a switch or router shall discard the packet or corrupt it as defined in Section 7.3, "Packet Receiver States," on page 146.

VCRC_check

good	VCRC check was good
bad	otherwise

ICRC_check

good	ICRC check was good
bad	otherwise

The link layer of a switch or router is only required to check ICRC on packets that are destined to that switch or router. On all other packets, a switch or router may omit the ICRC check by returning ICRC_check = good without checking the ICRC.

xport

IB	LNH indicates IB transport
raw	LNH indicates raw transport

lver_check

good	LVer equals 0x0
bad	otherwise

d_length_check

good	PktLen * 4 = received data bytes - 2 and (MTU +124)/4 >= PktLen > = minimum length
bad	otherwise

Received length is the number of bytes between the SDP and EGP. MTU is PortInfo.MTUCap. Minimum length is 5 for raw packets and 6 for IBA transport packets. See Section 7.7.8, "Packet Length (PktLen) - 11 bits," on page 167.

dlid_check

valid	for CAs: one of the following conditions is met: • DLID is a unicast LID of this CA, or • multicast LID configured for this CA, or • DLID is 0xFFFF (the permissive LID) and VL is 15 for switches and routers: DLID is not 0x0000.
invalid	otherwise

VL_check

valid	(VL is operational and PortState = Active) or (VL = 15 and DLID is unicast)
invalid	otherwise

VL15_check

valid	(VL <> 15) or (LNH indicates IBA local packet)
invalid	otherwise

buffer

avail	buffer is available for the packet
ovflow	otherwise

7.5 Link Packet Check

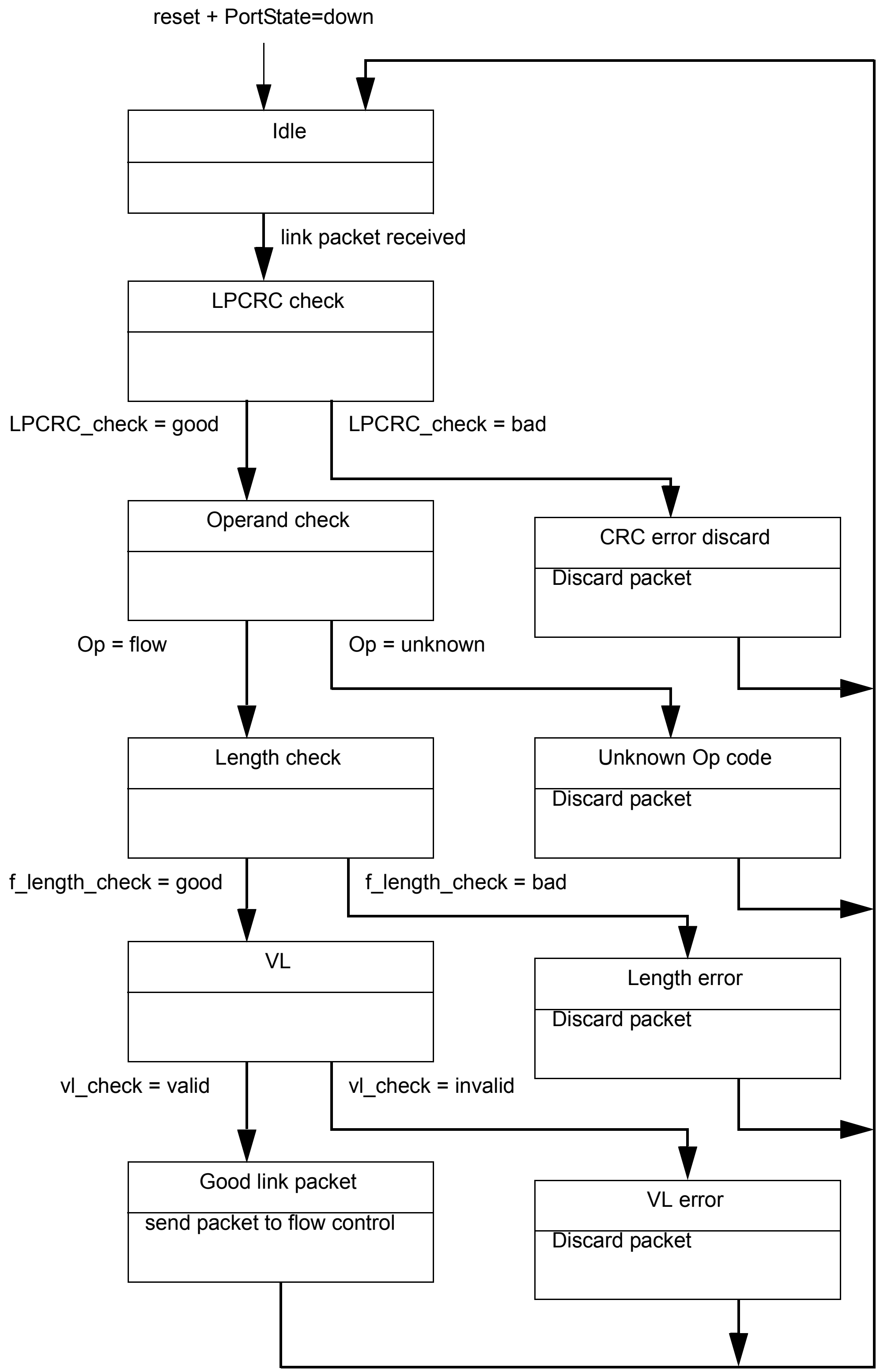
The only type of link packet currently defined is the flow control packet. See Section 7.9.4, "Flow Control Packet," on page 183.

C7-12: A port shall verify a link packet as specified by Figure 53 Link Packet Check machine on page 152 and Section 7.5, "Link Packet Check," on page 151 before passing it to the flow control.

LPCRC_check

good	LPCRC (Link Packet CRC) check was good
bad	otherwise

Figure 53 Link Packet Check machine



Op

flow	either flow control opcode is present
unknown	otherwise

VL_check

valid	(VL is supported or PortState is not Active) and VL is not 15
invalid	otherwise

During initialization, the number of active VLs may not have been configured yet, so receiving credits for a non-supported VL is only an error when in the active state.

f_length_check

good	length received = 6 bytes (including LPCRC)
bad	otherwise

7.6 VIRTUAL LANES MECHANISMS

Virtual lanes (VLs) provide a means to implement multiple logical flows over a single physical link. Link level flow control can be applied to one lane without affecting the others. Table 13 on page 153 summarizes the key attributes of VLs.

C7-13: An InfiniBand protocol aware device shall conform to the requirements defined by the rows labeled *required VLs*, *buffering*, and *ordering* in Table 13.

o7-2: An InfiniBand protocol aware device that implements more than one data VL shall conform to the requirements defined by the row labeled *flow control* in Table 13.

Table 13 Key Virtual Lane Characteristics

Attribute	Description
VL	Represents a logical flow over a given physical link.
VL Types	There are two types of VLs, one for normal traffic called a data VL and one reserved for subnet management traffic. The subnet management traffic VL is VL15. All other VLs are for normal traffic.

Table 13 Key Virtual Lane Characteristics

Attribute	Description
Required VLs	VL 15 shall be implemented in all IBA channel adapters, switches, and routers. VL 0 shall be implemented for application use in all IBA channel adapters, switches, and routers. VLs 1-14 may be implemented to support additional traffic segregation. If implemented, VLs shall be numbered as indicated in Table 14 VL Numbering and Interoperability on page 155
Buffering	Devices shall provide independent buffering resources for each VL. See 7.6.4 Buffering and Flow Control For Data VLs on page 156 for details.
Flow Control	Link-level flow control shall be implemented on a per VL basis. See 7.9 Flow Control on page 182 for description of flow control on data VLs. VL 15 does not use link-level flow control, however. See 7.6.3 Special VLs on page 155 for details. Flow control packets are not subject to flow control.
VL Field	4-bit field within the LRH indicating the actual VL being used by this packet.
SL Field	4-bit field located in the LRH indicating the requested service level within the local subnet. See 7.6.5 Service Level on page 158 for a description of this field.
Ordering	When fabric configuration is stable, unicast packets between the same source and destination LIDs within a subnet and using the same SL shall be ordered. Multicast packets shall also be similarly ordered. Note, however, that ordering is not guaranteed between unicast and multicast flows, even if on the same SL. Ordering is not maintained between different SLs. Packets on one SL may overtake packets on another SL, even if flowing through the same physical path within the fabric.

7.6.1 VL IDENTIFICATION

C7-14: The sending port of an InfiniBand protocol aware device shall identify each packet with the virtual lane to be used, this information being carried in the 4-bit VL field of link header. In addition, the local routing header shall contain a 4-bit Service Level (SL).

The use of the SL field is described in Section 7.6.5 on page 158.

7.6.2 NUMBER OF VLS SUPPORTED

C7-15: An InfiniBand protocol aware device shall conform to requirements defined by the rows labeled *VL numbering* and *configuration* in Table 14

C7-16: All ports of an Infiniband protocol aware device shall support VL15. Further, all ports shall support data VL0.

o7-3: Ports may support more than one data VL. If they do, they shall do in accordance with the allowed number specified in Table 14 on page 155.

C7-17: The data VLs shall be numbered sequentially starting from zero.

Thus, if an implementation supports 4 data VLs, they shall be numbered 0, 1, 2 and 3.

Table 14 VL Numbering and Inter-operability

Number of Data VLs Supported	VL Numbering	List of VL Configurations That Shall Be Supported[a]
1	VL0	1
2	VL0, VL1	2, 1
4	VL0 - VL3	4, 2, 1
8	VL0 - VL7	8, 4, 2, 1
15	VL0 - VL14	15, 8, 4, 2, 1

a. Because the port at the other end of the link may support a different number of VLs, the port must support operation with different numbers of VLs.

7.6.3 SPECIAL VLS

VL 15 is a special VL and must be supported by all ports. The following lists the properties of VL 15:

C7-18: VL15 shall not be subject to flow control (both link level and end-to-end), i.e. VL 15 packets may be transmitted at any time.

C7-19: Infiniband protocol aware devices shall discard VL15 packets if there is not enough room for reception. Other than the packet discard counter (16.1.3.5 PortCounters on page 763) this discard is done silently.

C7-20: All InfiniBand protocol aware devices shall support sourcing and sinking VL 15 packets.

C7-21: CAs and routers shall provide a minimum of a single packet buffer per port for VL15 on each port for reception.

C7-22: Switches shall provide a minimum of a single packet buffer for VL15 per switch.

C7-23: VL15 packets shall be scheduled preemptively, i.e. they are transmitted ahead of all other packets (including flow control packets).

C7-24: VL mapping in a switch does not apply to VL15. That is, a packet received by a switch on VL15 shall be transmitted on VL15 and no packet received on another VL shall be transmitted on VL15.

C7-25: The SL field shall be set to 0 by devices sourcing VL15 packets and ignored by devices checking and sinking VL15 packets.

C7-26: VL15 packets shall not be forwarded between subnets, i.e. they shall not have a GRH and they shall not be raw.

C7-27: Packets using VL15 shall have a maximum payload of 256 payload bytes.

7.6.4 BUFFERING AND FLOW CONTROL FOR DATA VLS

Virtual Lanes provide independent data streams on the same physical link.

For data VLs, separate guaranteed buffering resources, and separate flow control shall be provided. (For VL 15, different flow control and buffering restrictions apply, and are described in above.)

C7-28: For data VLs, each VL on each port shall provide the appearance of separate buffering resources, i.e. although dedicated buffering resources are not required, the ports must behave as if they were.

C7-29: Each port shall advertise the number of credits available for each data VL configured using flow control packets.

These credit packets and the flow control process are described in 7.9 Flow Control on page 182.

Table 15 Processing of Link Packets on page 157 details the behavior of a port when sending and receiving a link packet for a given data VL. The following terminology is used in this table (and elsewhere in this specification):

- A data VL is supported if its VL number is inside the range indicated by the PortInfo.VLCap attribute. This indicates that the data VL is supported by the port.
- A data VL is configured if its VL number is inside the range indicated by the PortInfo.OperationalVLs attribute.This indicates that the data VL is currently configured for use by the port.

(Refer to 14.2.5.6 PortInfo on page 665 for description of PortInfo.VLCap and PortInfo.OperationalVLs.)

C7-30: Each port shall send and receive link packets as specified in Table 15 Processing of Link Packets on page 157

Note, in this table, a required behavior has not been specified for the cases where the data VL is supported but not currently configured. This is done to support changing of the Data VL configuration. Note further, the Data VL configuration may be changed in any PortState including LInkActive.

Table 15 Processing of Link Packets

PortState	Status of a Data VL	Sending of Credits on that Data VL	Receiving of Credits on that Data VL
LinkInitialize	Data VL is Configured	Shall send link packets for that Data VL	Shall be accepted
	Data VL is supported but not currently configured	May send link packets for that Data VL	Shall be ignored, no error
	Data VL is not supported	Shall not send link packets for that Data VL	Shall be ignored, no error
LinkArm or LinkActive	Data VL is Configured	Shall send link packets on that Data VL	Shall be accepted
	Data VL is supported but not currently configured	Should not send link packets on that Data VL	Shall be ignored, no error
	Data VL is not supported	Shall not send link packets on unsupported data VLs	Shall be discarded, malformed packet reported

C7-31: Each port shall provide sufficient buffering for each configured data VL to be able to advertise credit for at least one packet with MTU payload.

Note, MTU payload here refers to the lesser of MTUCap and neighborMTU for that port

(See 7.7.8 Packet Length (PktLen) - 11 bits on page 167 for definition of the corresponding packet size requirement.)

C7-32: When a data packet arrives at a port, it shall be placed in the buffer associated with that input port and VL field in the packet.

7.6.5 SERVICE LEVEL

Service Level (SL) is used to identify different flows within an IBA subnet. It is carried in the local route header of the packet.

C7-33: The SL of a packet shall not be changed as a packet crosses the subnet.

The SL is an indication as to the service class of the packet. IBA does not assign any specific meaning to an SL value. SLs are intended as a mechanism to aid in providing differentiated services, improved fabric utilization and avoiding deadlock. However, the specifics on how this is done is beyond the scope of this specification.

The IBA specification does, however, define two mechanisms using SLs and VLs that are intended as tools to implement Quality of Service (QoS) related services. One is SL-to-VL mapping, the other is data VL arbitration. Both are described in detail below.

o7-4: If multiple data VLs are supported, then both SL-to-VL mapping and data VL arbitration must be supported (both described below).

If only a single data VL is supported, then neither are required (although SL-to-VL mapping may still be implemented for SL filtering--see 7.6.6 VL Mapping Within a Subnet on page 159 for a description of this).

C7-34: The only requirement for devices supporting only a single data VL with respect to SLs and VLs is that the device shall include the SL value in the SL field when sourcing a packet into an IBA subnet.

Note that switches are included in this list because they can be the source of packets via their SMI or GSI interfaces. Note also that this specification does not require the validation of SL field at the packet destination.

There are no ordering guarantees between packets of different SL.

The source for SL for different transport services is detailed in 9.10 Header and Data Field Source on page 386. For connected services (unreliable connected, reliable connected and reliable datagrams), the SL associated with the forward and reverse paths of the same connection may be different (i.e. on the same connection, the SL associated with the DeviceA:transmitWQ may be different from that for the DeviceB:transmitWQ). For unreliable and raw datagrams, however, a node can always respond to a datagram from some other node using the same SL as the original datagram.

The SL used for a given destination (DLID), QOS, partition etc. is ultimately provided by the subnet manager. It may also be from derived sources such as request packets, local management agents etc.

7.6.6 VL Mapping Within a Subnet

As a packet is routed across a subnet, it may be necessary for it to change VLs when it uses a given link. Examples of where this may be needed include:

1) The link may not support the VL previously used by the packet. This could happen when a device in the fabric supports a limited set of VLs.
2) Two traffic streams arriving on different input ports of a switch may be using the same outgoing link, and may also happen to be using the same VL when arriving at the switch. If VL mapping were not supported, then both traffic streams would have to use the same VL on the output port. VL mapping allows these two streams to be assigned different VLs on the outgoing links. In general, VL mapping offers greater flexibility in maintaining independent traffic flows within a fabric.

SL to VL mapping is used to change VLs as a packet crosses a subnet.

SL to VL mapping is required in channel adapters, switches, and routers that support more than one data VL. It is optional in those devices supporting only one data VL. If it is implemented it shall be implemented in accordance with the requirements of this section.

SL to VL mapping is done using a programmable mapping table. This is provided by the SLtoVLMappingTable.

o7-5: In channel adapters and routers that support SL to VL mapping, there shall be a logical table that maps the SL field in the packet LRH to the VL to be used for that output port. This table is 16 entries deep, with each port of the device having an independent table. All 16 possible values of SL shall be included in this table. The table indicates the VL number to be used by that packet when it is transmitted by the port.

o7-6: In switches that support SL to VL mapping, there shall be a logical table that maps the SL, input port and output port of the packet to the VL to be used for the next hop.

This table can be best viewed as a set of tables, one for each output port. Each of these per output port tables then indicates which VL should be used by the outgoing packet based on its SL field and the port that it arrived on. Because the switch supports an internal port (refer to 18.2.4.1 Switch Ports on page 849) that will also source packets that require VL mapping, this port is included as one of the input ports in the table.

o7-7: Thus, in switches that support SL to VL mapping, the overall SLtoVLMappingTable shall be 16*(num_ports+1)*num_ports deep, where num_ports is the number of external ports supported by the switch. All 16 possible values of SL shall be included in this table.

The table indicates the VL to be used by that packet for the next hop transmission based on packet SL, input port and output port.

This table provides mapping for the n+1 input ports (including the internal port) to n output ports.

Refer to Table 130 SLtoVLMappingTable on page 675 for details of on the SLtoVLMappingTable.

o7-8: Devices implementing SL to VL mapping shall behave as depicted in Table 16.

Table 16 SLtoVLMappingTable Behavior

VL Value in SLtoVLMappingTable	Action
VL15	Discard packet, no error.
Data VL not configured by port	Discard packet, no error.
Data VL configured by port	Forward packet to port using VL

The number of VLs supported is defined by the PortInfo.VLCap attribute, while the number configured is defined by the PortInfo.OperationalVLs attribute. (Refer to 14.2.5.6 PortInfo on page 665) for description of PortInfo.VLCap and PortInfo.OperationalVLs.)

Note, the SLtoVLMappingTable may be programmed with VL15 for any SL that is not authorized to use that port (for channel adapters and routers) or input-output port path (for switches). As indicated by the above table, packets are discarded if the SLtoVLMappingTable returns VL15. This filtering is intended as a mechanism to help protect against unauthorized use of SLs, and to help in breaking routing dependency loops (and thereby avoiding routing deadlocks).

7.6.7 INITIALIZATION AND CONFIGURATION

In order to allow devices to be built with different numbers of VLs, the SM must be able to configure the number of VLs to be used on a given link. The SM can query each port to determine the number of VLs it supports and then configure to a number supported by both ports on the link. Table 14 on page 155 depicts the number of VLs combinations that each device must support. The number of VLs supported is defined by the PortInfo.VLCap component while the number of VLs configured is defined by the PortInfo.OperationalVL (Refer to 14.2.5.6 PortInfo on page 665).

Ports may be configured to 1, 2, 4, 8 or 15 VLs and must be configured to a value equal to or less than the number supported. If an attempt is made to program the OperationalVLs to a value larger than the VLCap, the port may load OperationalVLs with any valid value.

A port must be configured with the same number of VLs for both its sending and receiving directions.

Modification of the SLtoVLMappingTable may be made while the port is in operation.

o7-9: If a port implements SL-to-VL mapping, it shall not allow any packet in transit to be fragmented as a result of changing the SLtoVLMappingTable contents.

Packets may be discarded or mis-mapped during this change, however.

When a channel adapter, router, or switch initializes, the SLtoVLMappingTable is not required to be initialized (i.e.the contents are undefined). The table should be initialized by the SM prior to use by data traffic.

7.6.8 VL SCHEDULING AND FLOW CONTROL FOR VL15 AND FLOW CONTROL PACKETS

VL15 (i.e. subnet management packets) traffic and flow control packets will use preemptive scheduling. The order of precedence is depicted in Table 17.

7.6.9 VL ARBITRATION AND PRIORITIZATION

VL arbitration refers to the arbitration done for an outgoing link on a switch, router or channel adapter. Each output port has a separate arbiter. The arbiter selects the next packet to transmit from the set of candidate packets available for transmission on that port.

C7-35: The arbiter shall not violate packet ordering rules, i.e. packets on a given VL shall not be reordered.

The following describes the algorithm to be used by the VL arbiter.

7.6.9.1 VL ARBITRATION WHEN ONLY ONE DATA VL IS IMPLEMENTED

Table 17 depicts the arbitration rules for switch, router or channel adapters that implement only a single data VL. This is a simple priority scheme

Table 17 Arbitration Rules for Devices with only one data VL

Packet type	Precedence order
VL15	Highest
Flow control packet	2nd highest
VL0	Lowest

where all packets at a precedence level are sent before any packets at a lower precedence level.

o7-10: Devices implementing only a single data VL shall transmit packets on its output ports using the arbitration rules depicted in Table 17 Arbitration Rules for Devices with only one data VL on page 162.

7.6.9.2 VL ARBITRATION WHEN MULTIPLE DATA VL S ARE IMPLEMENTED

The implementation of multiple data VLs is an optional feature in IBA. If they are implemented, however, the implementation shall conform to the specification detailed in this section.

o7-11: For devices implementing more than one data VL, the transmission of VL15 packets and flow control packets shall be the same as depicted in Table 17 on page 162 except that here all the data VLs are at a lower priority than VL15 (highest) and flow control packets (second highest).

o7-12: Devices implementing more than one data VL shall also implement the algorithm described in Section 7.6.9.2 for arbitrating between packets on the data VLs.

A two level scheme is employed, using preemptive scheduling layered on top of a weighted fair scheme. Additionally, the scheme provides a method to ensure forward progress on the low-priority VLs. The weighting, prioritization, and minimum forward progress bandwidth is programmable.

VL arbitration is controlled by the VLArbitrationTable (refer to 14.2.5.9 VLArbitrationTable on page 675). This table shall consist of three components, *High-Priority*, *Low-Priority* and *Limit of High-Priority*. The *High-Priority* and *Low-Priority* components are each a list of VL/Weight pairs. The *High-Priority* list shall have a minimum length of one and a maximum of length of 64. The *Low-Priority* list shall have a minimum length equal to the number of data VLs supported and a maximum of length of 64. The

High-Priority and *Low-Priority* component lists are allowed to be of different length.

Each list entry shall contain (1) a VL number (values from 0-14), and (2) a weighting value (values 0-255), indicating the number of 64 byte units which may be transmitted from that VL when its turn in the arbitration occurs. The PktLen field in the LRH is used to determine the number of units in the packet. (Note, the VCRC and also the symbols between packets introduced by the physical layer should not be included in VL arbitration weight calculations.) The calculation shall be maintained to 4 byte increments.

A weight of 0 indicates that this entry should be skipped.

If a list entry is programmed for VL15 or for a VL that is not supported or is not currently configured by the port, the port may either skip that entry or send from any supported VL for that entry.

Note, that the same data VL may be listed multiple times in the High or Low-Priority component list, and, further, it can be listed in both lists.

Each configured data VL should be listed in at least one of the component lists. There is, however, no requirement for a device to check for this case. Should a configured data VL not appear in either component list, packets for this data VL may be dropped, sent when the arbiter has no packets to send or never sent.

The *Limit of High-Priority* component indicates the amount high-priority packets that can be transmitted without an opportunity to send a low priority packet. Specifically, the number of bytes that can be sent is *Limit of High-Priority* times 4K bytes, with the counting done the same as described above for weights (i.e. the calculation is done to 4 byte increments and a High-Priority packet can be sent if current byte count has not exceed exceeded the *Limit of High-Priority*). A value of 255 indicates that the byte limit is unbounded. (Note, it the 255 value is used, forward progress of low priority packets is not guaranteed by this arbitration scheme.) A value of 0 indicates that only a single packet from the high-priority table may be sent before an opportunity is given to the low-priority table.

The VLArbitrationTable may be modified when the port is active. This modification shall not result in fragmentation of any packet that is in transit. Arbitration rules may violated during this change, however.

When a channel adapter, router, or switch initializes, the VLArbitrationTable is not required to be initialized (i.e.the contents are undefined). The table should be initialized by the SM prior to use by data traffic.

7.6.9.2.1 ARBITRATION RULES BETWEEN VL15, LINK CONTROL AND DATA VL PACKETS

The rules of table Table 17 on page 162 apply, where the data VLs (VL0-VL14) have the lowest priority.

7.6.9.2.2 ARBITRATION RULES FOR DATA VL PACKETS

When there are no VL15 or Flow Control packets to send, the arbitration rules in this section apply.

7.6.9.2.3 ARBITRATION RULES BETWEEN HIGH AND LOW PRIORITY COMPONENTS

The *High-Priority* and *Low-Priority* components form a two level priority scheme. Each of these components (or tables) may have a packet available for transmission. A packet is available for transmission from the High Priority table if the following test succeeds:

For each entry in the High Priority table, determine if:

1) the VL field matches that of any packets that are currently waiting for transmission for this port AND
3) there is available credit to send that packet

An entry with 0 weight is considered not in the list.

Note, Implementations may check if HiPriAvailWeight is available in determining if a packet is available.

Upon completion of transmission of a packet the following test should be done to determine which table to use to transmit the next packet:

If the High-Priority table has an available packet for transmission (as defined above) and the HighPriCounter has not expired, then the High-Priority is said to be *active* and a packet may be sent from the High-Priority table.

If the High-Priority table does not have an available packet for transmission (as defined above), or if the HighPriCounter has expired, then the HighPriCounter shall be reset, the Low-Priority table is said to be *active* and a packet may be sent from the Low-Priority table.

The following rules govern the operation of the HighPriCounter:

1) The HighPriCounter expires when its current value is negative.
2) If the value in the *Limit of High-Priority* component is not 255, then for each High-Priority packet transmitted, the size of the packet (as defined by the PktLen field in the LRH) is deducted from the current value of the HighPriCounter. The calculation should be maintained to 4 byte increments.
3) When the HighPriCounter is reset, the value in the *Limit of High-Priority* component times 4K bytes is loaded into the HighPriCounter.

7.6.9.2.4 ARBITRATION RULES WITHIN THE HIGH AND LOW COMPONENTS

Within each High or Low Priority table, weighted fair arbitration is used, with the order of entries in each table specifying the order of VL scheduling, and the weighting value specifying the amount of bandwidth allocated to that entry. Each entry in the table is processed in order.

A separate pointer and available weight count is maintained for each of the two tables. The pointers identify the current entry in the table, while the available weight count indicates the amount of weight that the current entry has available for data packet transmission. When a table is active (as defined in the previous section), the current entry in the table is inspected. A packet corresponding to this entry will be sent to the output port for transmission and the packet size (in 4 byte increments) will be deducted from the available weight count for the current entry, if all of the following are true:

1) The available weight for the list entry is positive.
2) There is a packet available for the VL of the entry
3) Buffer credit is available for this packet.

Note, if the available weight at the start of a new packet is positive, condition 1 above is satisfied, even if the packet is larger than the available weight.

When any of these conditions is not true, the next entry in the table is inspected. The current pointer is moved to the next entry in the table, the available weight count is set to the weighting value for this new entry, and the above test repeated. This is repeated until a packet is found that can be sent to the port for transmission. If the entire table is checked and no entry can be found satisfying the above criteria, the other table becomes active.

This description depicts the logical flow of the arbitration process, but does not specify performance requirements. Implementations shall perform in a logically consistent manner with the above description. Implementations may process steps in parallel and may pipeline tests. As an example of pipelining of tests, the check that there be available packets may return false if a packet has just recently been forwarded to output port but the arbiter logic has not processed its arrival.

Further, implementations are not required to implement the pointers, available weight counter and HighPriCounter. They must, however, behave in a manner equivalent to that described in this section.

7.7 LOCAL ROUTE HEADER

Local Routing Header - LRH - 8 bytes

The Local Routing Header (LRH) contains the fields for local routing by switches within a IBA subnet. The LRH is at the start of every packet and the packet ends with the Variant CRC. The LRH is 8 bytes long. For additional information on overall packet layout, see Chapter 5: Data Packet Format on page 124.

Figure 54 Local Route Header (LRH)

bits / bytes	31-24		23-16			15-8	7-0
0-3	VL	LVer	SL	Rsv2	LNH	Destination Local Identifier	
4-7	Reserve 5	Packet Length (11 bits)				Source Local Identifier	

C7-36: The LRH shall use the format specified in Figure 54 Local Route Header (LRH) on page 166.

7.7.1 VIRTUAL LANE (VL) - 4 BITS

Specifies a virtual lane to be used for a packet. This field identifies which receive buffer and which receive flow control credits should be used for the received packet.

C7-37: The VL field shall be set to the VL on which the packet is sent.

The virtual lane can change from link to link in a subnet. Since the Virtual Lane can change, the Link Virtual Lane is not included in the Invariant CRC field.

7.7.2 LINK VERSION (LVER) - 4 BITS

Specifies the version of the Local Routing Header used for this packet. This version applies to the general packet structure including the LRH fields and the variant CRC.

C7-38: The LVer field shall be set to 0x0.

If a receiving device does not support the Link Version specified then the packet is discarded.

7.7.3 SERVICE LEVEL (SL) - 4 BITS

The Service Level field. This field is used by switches to determine the Virtual Lane used for this packet. This is described in Section 7.6.5 on page 158.

7.7.4 RESERVE - 2 BITS

C7-39: The 2-bit Reserve field shall be transmitted as 00 and shall be ignored on receive.

7.7.5 LINK NEXT HEADER (LNH) - 2 BITS

Specifies what headers following the Local Routing Header. The first bit (msb) indicates whether the packet uses IBA transport. The second bit (lsb) indicates whether a GRH/IPv6 header is present.

Table 18 Link Next Header Definition

Packet Type	LNH bit 1 IBA Transport	LNH bit 0 GRH/IPv6 header	Transport	Next Header
IBA global	1	1	IBA	GRH
IBA local	1	0	IBA	BTH
IP - non-IBA transport	0	1	Raw	IPv6
Raw	0	0	Raw	RWH (Ethertype)

C7-40: The LNH field shall indicate the packet type of the following packet as defined by Table 18 Link Next Header Definition on page 167.

7.7.6 DESTINATION LOCAL IDENTIFIER (DLID) - 16 BITS

Specifies the LID of the port to which the subnet delivers the packet. LIDs are unique within a subnet. More specifically it identifies the route to take to the destination port. If the packet is to be routed to another subnet, then this is the LID of the Router.

7.7.7 RESERVE - 5 BITS

C7-41: The 5 bit reserve field shall be transmitted as 00000 and shall be ignored on receive.

7.7.8 PACKET LENGTH (PKTLEN) - 11 BITS

The number of 4 byte words contained in the packet.

C7-42: The value of the PktLen field shall equal the number of bytes in all the fields starting with the first byte of the Local Route Header and the last byte before the Variant CRC, inclusive, divided by 4.

The maximum allowable size of all headers plus the CRC fields is 126 bytes. The maximum value of this field is (4096 + 126 - 2)/4 = 4220 / 4=

1055, reflecting a maximum of 126 bytes for all headers and CRCs minus the uncounted variant CRC.

Table 19 Packet Size

MTU	Maximum Packet Length (Bytes/4)	Maximum Bytes (MTU+126)
256	95	382
512	159	638
1024	287	1150
2048	543	2174
4096	1055	4222

C7-43: For packets with IBA transport, the smallest allowed value for Packet Length is 6 (24 Bytes) including LRH.

C7-44: For raw packets, the smallest allowed value for Packet Length is 5 (20 Bytes) including LRH.

C7-45: The maximum allowed value for Packet Length is the value shown in Table 19 Packet Size on page 168 for the smaller of MTUCap and NeighborMTU.

7.7.9 Source Local Identifier (SLID) - 16 bits

C7-46: For all non-directed route packets, the SLID shall be a LID of the port which injected the packet onto the subnet.

For requirements on DLID in directed route packets, see 14.2 Subnet Management Class on page 642.

The subnet manager assigns each node a LID which is unique within a subnet.

7.8 CRCs

7.8.1 Invariant CRC (ICRC) - 4 Bytes

Specifies a Cyclic Redundancy Code covering all the fields of the Packet which are invariant from end to end through all switches and routers on the network. This field is present in all IBA packets but is NOT present in Raw Packets because for raw packets it is not known which fields will be invariant. The CRC calculation is re-started with each packet in the message. Which header fields that are included depends on whether the Global Routing Header is present because the router may modify additional header fields.

C7-47: The ICRC field shall be present in all IBA transport packets.

C7-48: The ICRC field shall be calculated as specified in Section 7.8.1, "Invariant CRC (ICRC) - 4 Bytes," on page 168.

If the packet is local to the subnet (the Global Routing Header is not present), then the ICRC calculation is as follows:

- With no GRH, the ICRC includes:
  - Local Routing Header: except for the VL.
  - Base Transport Header: except for the Resv8a field
  - Extension Transport Headers (if present),
  - Packet Payload (if present),
- With no GRH, the ICRC excludes: (these fields are replaced with 1s for the ICRC calculation)
  - Local Routing Header: VL,
  - Base Transport Header: Resv8a.

If the packet is routed between subnets, so the Global route header is present, the ICRC calculation is as follows:

- With a GRH, the ICRC includes:
  - Global Routing Header: Version, Payload length, Next Header, Source IPV6 address, and Destination IPV6 address
  - Base Transport Header, except for the Resv8a field,
  - Extension Transport Headers (if present),
  - Packet Payload (if present).
- With a GRH, the ICRC excludes: (these fields are replaced with 1's for the CRC calculation)
  - Local Routing Header, all fields,
  - Global Routing Header: Flow label, Traffic Class, and Hop Limit fields.
  - Base Transport Header: Resv8a.

All fields in the packet. including those excluded from the Invariant CRC, are protected by the Variant CRC described in the next section.

The polynomial used is the same CRC-32 used by Ethernet - 0x04C11DB7. The procedure for the calculation is:

1) The initial value of the CRC-32 calculation is 0xFFFFFFFF.
2) The CRC calculation is done in big endian byte order with the least significant bit of the most significant byte being the first bits in the CRC calculation.

3) The bit sequence from the calculation is complemented and the result is the ICRC.

4) The resulting bits are sent in order from the bit representing the coefficient of the highest term of the remainder polynomial. The least significant bit, most significant byte first ordering of the packet does not apply to the ICRC field.

The CRC always starts with LRH:LVer bit 0, whether GRH is present or not.

This bit and byte ordering is consistent with Ethernet's CRC calculation.

Figure 55 CRC Calculation Order

bits bytes	31-24	23-16	15-8	7-0
	Bit0 in CRC Calculation, Bit 0, Byte 0 ↓			
0-3	Byte0	Byte 1	Byte 2	Byte3
4-7	Byte 4	Byte 5	Byte 6	Byte 7
8-11	Byte 8	Byte 9	Byte 10	Byte 11
...				

7.8.2 VARIANT CRC (VCRC) - 2 BYTES

Specifies a Cyclic Redundancy Code covering all fields of the Packet. This field is present in all data packets including Raw Packets and includes all bytes from the first byte of the LRH to the last byte before the Variant CRC, inclusive. Since a number of these fields can change as the packet is processed by switches and routers the Variant CRC may have to regenerated at each Link through the subnet. If a switch does not change any fields including the Link Virtual Lane, then the Variant CRC does not have to be regenerated.

C7-49: The VCRC field shall be present in all data packets.

C7-50: The VCRC field shall be calculated as specified in Section 7.8.2, "Variant CRC (VCRC) - 2 Bytes," on page 170.

The polynomial used is the same CRC-16 used by HIPPI-6400 - 0x100B. The procedure for the calculation is:

1) The initial value of the CRC-16 calculation is 0xFFFF.

2) The CRC calculation is done in big endian byte order with the least significant bit of the first byte of the Local Route Header (bit 0 of LRH:LVer) being the first bit in the CRC calculation.

3) The bit sequence from the calculation is complemented and the result is the VCRC.

4) The resulting bits are sent in order from the bit representing the coefficient of the highest term of the remainder polynomial. The least significant bit, most significant byte first ordering of the packet does not apply to the VCRC field.

This bit and byte ordering is consistent with Ethernet's CRC calculation.

7.8.3 Link Packet CRC (LPCRC) - 2 Bytes

Specifies a Cyclic Redundancy Code covering all fields of the Link Packet. This field is present in all Link packets including Flow Control Link Packets and includes all bytes from the first byte of the Opcode to the last byte before the LPCRC, inclusive. This field is always computed for each Link-packet.

C7-51: The LPCRC field shall be present in all link packets.

C7-52: The LPCRC field shall be calculated as specified in Section 7.8.3, "Link Packet CRC (LPCRC) - 2 Bytes," on page 171.

The polynomial used is the same CRC-16 used by Variant CRC and HIPPI-6400 - 0x100B. The procedure for the calculation is:

1) The initial value of the CRC-16 calculation is 0xFFFF.

2) The CRC calculation is done in big endian byte order with the least significant bit of the first byte of the Local Route Header (bit 0 of LRH:LVer) being the first bit in the CRC calculation.

3) The bit sequence from the calculation is complemented and the result is the LPCRC.;

4) The resulting bits are sent in order from the bit representing the coefficient of the highest term of the remainder polynomial. The least significant bit, most significant byte first ordering of the packet does not apply to the LPCRC field.

This bit and byte ordering is consistent with Ethernet's CRC calculation.

7.8.4 CRC Calculation Samples

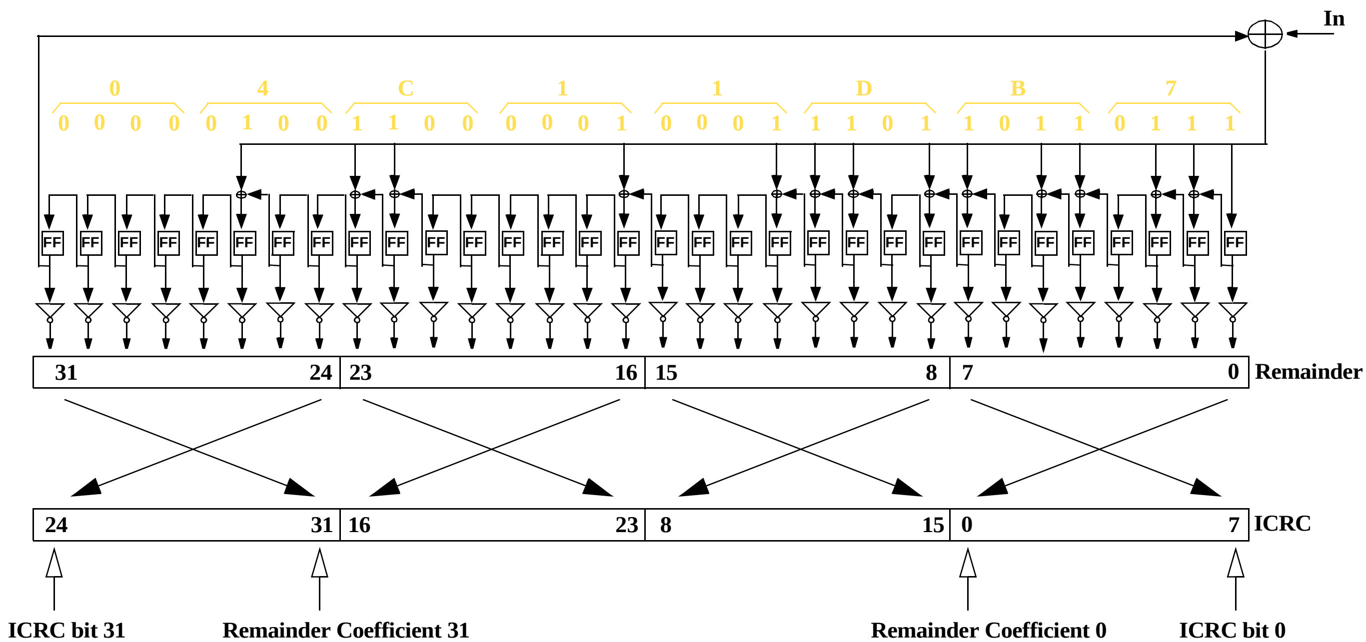
The following is an example of CRC calculation. The requirements for the CRC calculation are specified above, this section is intended for informative purposes only.

7.8.4.1 ICRC Generator

The polynomial used for ICRC calculation is 0x04C11DB7. The seed value is 0xFFFFFFFF. The ICRC Generator Remainder is the complement of the resulting calculation.

The ICRC Generator actual implementation is not specified. The diagram in Figure 56 is provided as a reference with the sole purpose of clarifying the calculation details and does not imply a required implementation.

Figure 56 ICRC Generator



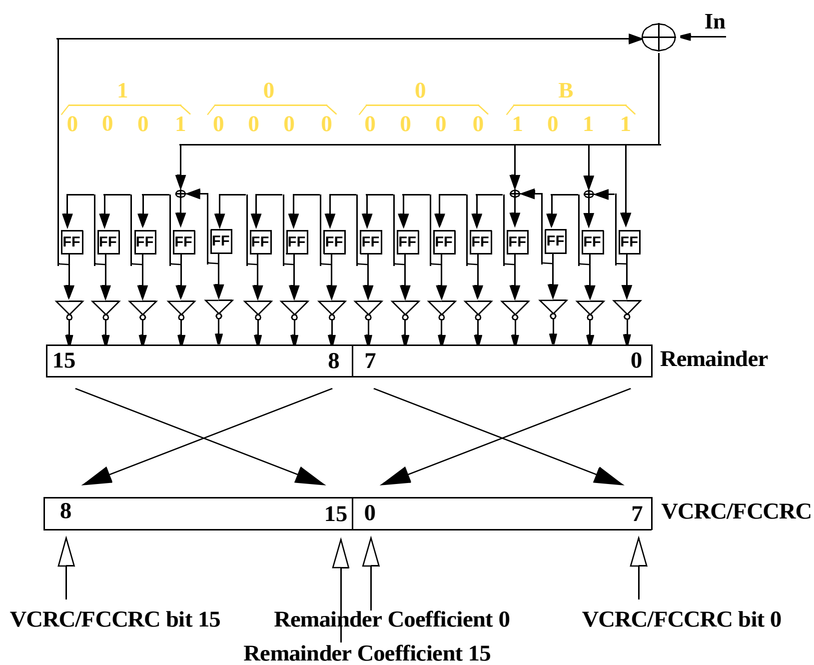
The 32 Flip-Flops are initialized to 1 before every ICRC generation. The packet is fed to the reference design of Figure 56 one bit at a time in the order specified in Section 7.8.1 on page 168. The **Remainder** is the ***bit-wise NOT*** of the value stored at the 32 Flip-Flops after the last bit of the packet was clocked into the ICRC Generator. **ICRC Field** is obtained from the **Remainder** as shown in Figure 56. **ICRC Field** is transmitted using Big Endian byte ordering like every field of an InfiniBand packet.

7.8.4.2 VCRC GENERATOR

The polynomial used for VCRC and FCCRC calculation is 0x100B. The seed value is 0xFFFF. The VCRC/FCCRC Generator Remainder is the complement of the resulting calculation.

The VCRC and FCCRC are generated in the same manner as described above for the ICRC. Figure 57 shows the reference design for the VCRC / FCCRC Generator.

Figure 57 VCRC / FCCRC Generator



7.8.4.3 SAMPLE PACKETS

7.8.4.3.1 LOCAL PACKET EXAMPLE

Figure 58 shows the structure of the local packet used for the example. The packet is a ***RDMA Write Only*** carrying a payload of 14 bytes.

Figure 58 Local Packet Example

The header values for the sample packet are shown in Table 20, Table 21 and Table 22 respectively. The data payload is shown in Table 23

.

Table 20 LRH

Field	Value
VL	0x7
LVer	0x0
SL	0x1
LNH	0x2
DLID	0x375C
PktLen	0xE
SLID	0x17D2

Table 21 BTH

Field	Value
Opcode	0x0A
SE	0x0
M	0x0
Pad	0x2
TVer	0x0
PKey	0x2487
Dest QP	0x87B1B3
AckReq	0x0
PSN	0x0DEC2A

Table 22 RETH

Field	Value
VA	0x01710A1C015D4002
RKey	0x38f27A05

Table 22 RETH

Field	Value
DMA Length	0x0000000E

Table 23 Payload

Byte	Value
0	0xBB
1	0x88
2	0x4D
3	0x85
4	0xFD
5	0x5C
6	0xFB
7	0xA4
8	0x72
9	0x8B
10	0xC0
11	0x69
12	0x0E
13	0xD4

The combined byte stream for the Local Packet (before ICRC and VCRC) is shown in Table 24

Table 24 Local Packet Byte Stream (before ICRC and VCRC)

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	15	0xB3	30	0x7A	45	0x8B
1	0x12	16	0x00	31	0x05	46	0xC0
2	0x37	17	0x0D	32	0x00	47	0x69
3	0x5C	18	0xEC	33	0x00	48	0x0E
4	0x00	19	0x2A	34	0x00	49	0xD4
5	0x0E	20	0x01	35	0x0E	50	0x00
6	0x17	21	0x71	36	0xBB	51	0x00
7	0xD2	22	0x0A	37	0x88		
8	0x0A	23	0x1C	38	0x4D		
9	0x20	24	0x01	39	0x85		
10	0x24	25	0x5D	40	0xFD		
11	0x87	26	0x40	41	0x5C		
12	0x00	27	0x02	42	0xFB		
13	0x87	28	0x38	43	0xA4		
14	0xB1	29	0xF2	44	0x72		

Table 25 shows the masked byte stream used for ICRC calculation.

Table 25 Masked Byte Stream for ICRC Calculation

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x**F**0	15	0xB3	30	0x7A	45	0x8B
1	0x12	16	0x00	31	0x05	46	0xC0
2	0x37	17	0x0D	32	0x00	47	0x69
3	0x5C	18	0xEC	33	0x00	48	0x0E
4	0x00	19	0x2A	34	0x00	49	0xD4
5	0x0E	20	0x01	35	0x0E	50	0x00
6	0x17	21	0x71	36	0xBB	51	0x00
7	0xD2	22	0x0A	37	0x88		
8	0x0A	23	0x1C	38	0x4D		
9	0x20	24	0x01	39	0x85		
10	0x24	25	0x5D	40	0xFD		
11	0x87	26	0x40	41	0x5C		
12	0xFF	27	0x02	42	0xFB		
13	0x87	28	0x38	43	0xA4		
14	0xB1	29	0xF2	44	0x72		

Generated ICRC is: 0x9625B75A

Generated VCRC is: 0x45FA

Table 26 shows the complete Local Packet Byte Stream.

Table 26 Local Packet Byte Stream

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	15	0xB3	30	0x7A	45	0x8B
1	0x12	16	0x00	31	0x05	46	0xC0
2	0x37	17	0x0D	32	0x00	47	0x69
3	0x5C	18	0xEC	33	0x00	48	0x0E
4	0x00	19	0x2A	34	0x00	49	0xD4
5	0x0E	20	0x01	35	0x0E	50	0x00
6	0x17	21	0x71	36	0xBB	51	0x00
7	0xD2	22	0x0A	37	0x88	52	0x96
8	0x0A	23	0x1C	38	0x4D	53	0x25
9	0x20	24	0x01	39	0x85	54	0xB7
10	0x24	25	0x5D	40	0xFD	55	0x5A
11	0x87	26	0x40	41	0x5C	56	0x45
12	0x00	27	0x02	42	0xFB	57	0xFA
13	0x87	28	0x38	43	0xA4		
14	0xB1	29	0xF2	44	0x72		

7.8.4.3.2 GLOBAL PACKET EXAMPLE

Figure 59 shows the structure of the Global packet used for the example.

Figure 59 Global Packet Example

LRH	GRH	BTH	RETH	Data Payload	ICRC	VCRC

The BTH, RETH and data payload for the Global example packet are the same as for the Local packet one. The values for the LRH and GRH fields are shown in Table 27 and Table 28.

.

Table 27 LRH

Field	Value
VL	0x7
LVer	0x0
SL	0x1
LNH	0x3
DLID	0x375C
PktLen	0x18
SLID	0x17D2

Table 28 GRH

Field	Value
IPVer	0x6
TClass	0x00
FlowLabel	0x00000
PayLen	0x0030
NxtHdr	0x00
HopLmt	0x10
SGID	0x00000000000001250000000000000026
DGID	0x00000000000001170000000000000096

The combined byte stream for the Global Packet (before ICRC and VCRC) is shown in Table 29

Table 29 Global Packet Byte Stream (before ICRC and VCRC)

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	25	0x00	50	0x24	75	0x0E
1	0x13	26	0x00	51	0x87	76	0xBB
2	0x37	27	0x00	52	0x00	77	0x88
3	0x5C	28	0x00	53	0x87	78	0x4D
4	0x00	29	0x00	54	0xB1	79	0x85
5	0x18	30	0x00	55	0xB3	80	0xFD
6	0x17	31	0x26	56	0x00	81	0x5C
7	0xD2	32	0x00	57	0x0D	82	0xFB
8	0x60	33	0x00	58	0xEC	83	0xA4
9	0x00	34	0x00	59	0x2A	84	0x72
10	0x00	35	0x00	60	0x01	85	0x8B
11	0x00	36	0x00	61	0x71	86	0xC0
12	0x00	37	0x00	62	0x0A	87	0x69
13	0x30	38	0x01	63	0x1C	88	0x0E
14	0x00	39	0x17	64	0x01	89	0xD4
15	0x10	40	0x00	65	0x5D	90	0x00
16	0x00	41	0x00	66	0x40	91	0x00
17	0x00	42	0x00	67	0x02		
18	0x00	43	0x00	68	0x38		
19	0x00	44	0x00	69	0xF2		
20	0x00	45	0x00	70	0x7A		
21	0x00	46	0x00	71	0x05		
22	0x01	47	0x96	72	0x00		
23	0x25	48	0x0A	73	0x00		
24	0x00	49	0x20	74	0x00		

Table 30 shows the masked byte stream used for ICRC calculation.

Table 30 Masked Byte Stream for ICRC Calculation

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x**FF**	25	0x00	50	0x24	75	0x0E
1	0x**FF**	26	0x00	51	0x87	76	0xBB
2	0x**FF**	27	0x00	52	0xFF	77	0x88
3	0x**FF**	28	0x00	53	0x87	78	0x4D
4	0x**FF**	29	0x00	54	0xB1	79	0x85
5	0x**FF**	30	0x00	55	0xB3	80	0xFD
6	0x**FF**	31	0x26	56	0x00	81	0x5C
7	0x**FF**	32	0x00	57	0x0D	82	0xFB
8	0x6**F**	33	0x00	58	0xEC	83	0xA4
9	0x**FF**	34	0x00	59	0x2A	84	0x72
10	0x**FF**	35	0x00	60	0x01	85	0x8B
11	0x**FF**	36	0x00	61	0x71	86	0xC0
12	0x00	37	0x00	62	0x0A	87	0x69
13	0x30	38	0x01	63	0x1C	88	0x0E
14	0x00	39	0x17	64	0x01	89	0xD4
15	0x**FF**	40	0x00	65	0x5D	90	0x00
16	0x00	41	0x00	66	0x40	91	0x00
17	0x00	42	0x00	67	0x02		
18	0x00	43	0x00	68	0x38		
19	0x00	44	0x00	69	0xF2		
20	0x00	45	0x00	70	0x7A		
21	0x00	46	0x00	71	0x05		
22	0x01	47	0x96	72	0x00		
23	0x25	48	0x0A	73	0x00		
24	0x00	49	0x20	74	0x00		

ICRC Result is:0xB67D1ED1

VCRC Result is: 0xB148

Table 31 shows the complete Global Packet Byte Stream.

Table 31 Global Packet Byte Stream

Byte	Value	Byte	Value	Byte	Value	Byte	Value
0	0x70	25	0x00	50	0x24	75	0x0E
1	0x13	26	0x00	51	0x87	76	0xBB
2	0x37	27	0x00	52	0x00	77	0x88
3	0x5C	28	0x00	53	0x87	78	0x4D
4	0x00	29	0x00	54	0xB1	79	0x85
5	0x18	30	0x00	55	0xB3	80	0xFD
6	0x17	31	0x26	56	0x00	81	0x5C
7	0xD2	32	0x00	57	0x0D	82	0xFB
8	0x60	33	0x00	58	0xEC	83	0xA4
9	0x00	34	0x00	59	0x2A	84	0x72
10	0x00	35	0x00	60	0x01	85	0x8B
11	0x00	36	0x00	61	0x71	86	0xC0
12	0x00	37	0x00	62	0x0A	87	0x69
13	0x30	38	0x01	63	0x1C	88	0x0E
14	0x00	39	0x17	64	0x01	89	0xD4
15	0x10	40	0x00	65	0x5D	90	0x00
16	0x00	41	0x00	66	0x40	91	0x00
17	0x00	42	0x00	67	0x02	92	0xB6
18	0x00	43	0x00	68	0x38	93	0x7D
19	0x00	44	0x00	69	0xF2	94	0x1E
20	0x00	45	0x00	70	0x7A	95	0xD1
21	0x00	46	0x00	71	0x05	96	0xB1
22	0x01	47	0x96	72	0x00	97	0x48
23	0x25	48	0x0A	73	0x00		
24	0x00	49	0x20	74	0x00		

7.8.4.3.3 LINK PACKET EXAMPLE

The field values for the Link Packet example are shown in Table 32.

Table 32 Link Packet

Field	Value
Op	0x0
FCTBS	0x10D
VL	0x5

Table 32 Link Packet

Field	Value
FCCL	0x21B

Generated FCCRC: 0xF9C9

Table 33 Link Packet Byte Stream

Byte	Value
0	0x01
1	0x0D
2	0x52
3	0x1B
4	0xF9
5	0xC9

7.9 FLOW CONTROL

7.9.1 INTRODUCTION

This section describes the link level flow control mechanism utilized by IBA to prevent the loss of packets due to buffer overflow by the receiver at each end of a link. This mechanism does not describe end to end flow control such as might be utilized to prevent transmission of messages during periods when receive buffers are not posted.

Throughout this section, the terms “transmitter” and “receiver” are utilized to describe each end of a given link. The transmitter is the node sourcing data packets. The receiver is the consumer of the data packets. Each end of the link has a transmitter and a receiver.

IBA utilizes an “absolute” credit based flow control scheme. Unlike many traditional flow control schemes which provide incremental updates that are added to the transmitters available buffer pool, IBA receivers provide a “credit limit”. A credit limit is an indication of the total amount of data that the transmitter has been authorized to send since link initialization.

Errors in transmission, in data packets, or in the exchange of flow control information can result in inconsistencies in the flow control state perceived by the transmitter and receiver. The IBA flow control mechanism provides for recovery from this condition. The transmitter periodically sends an indication of the total amount of data that it has sent since link initialization. The receiver uses this data to re-synchronize the state between the receiver and transmitter.

7.9.2 Flow Control Blocks

The term “flow control block”, or simply “block” indicates a quantity of data in a data packet. This quantity is defined to be the size of the data packet in bytes (every byte between the local route header and the variant CRC, inclusive) divided by 64 bytes, and rounded up to the next integral value.

7.9.3 Relationship to Virtual Lanes

The flow control algorithm defined in this chapter is applied to each virtual lane independently, except for virtual lane 15 which is not subject to link level flow control.

7.9.4 Flow Control Packet

Figure 60 Flow Control Packet Format

<table>
<tr><th colspan="7">Flow Control Packet - general format</th></tr>
<tr><th>bits
bytes</th><th colspan="2">31-24</th><th>23-16</th><th colspan="2">15-8</th><th>7-0</th></tr>
<tr><td>0-3</td><td>Op</td><td colspan="2">FCTBS</td><td>VL</td><td colspan="2">FCCL</td></tr>
<tr><td>4-5</td><td colspan="3">LPCRC</td><td colspan="3"></td></tr>
</table>

C7-53: Flow control packets shall be sent for each VL except VL15 upon entering the LinkInitialize state. When in the PortStates LinkInitialize, LinkArm or LinkActive, a flow control packet for a given virtual lane shall be transmitted prior to the passing of 65,536 symbol times since the last time a flow control packet for the given virtual lane was transmitted.

C7-54: Flow control packets shall use the format specified in Figure 60 Flow Control Packet Format on page 183.

A symbol time is defined as the time required to transmit an eight bit data quantity onto the link. Flow control packets may be transmitted as often as necessary to return credits and enable efficient utilization of the link. See Section 7.6.4, “Buffering and Flow Control For Data VLs,” on page 156 for additional information.

7.9.4.1 Flow Control Packet Fields

7.9.4.1.1 Operand (Op) - 4 Bits

The flow control packet is a link packet with one of two Op (operand) values: An operand of 0x0 indicates a normal flow control packet. An operand value of 0x1 indicates a flow control init packet.

C7-55: When in the PortState LinkInitialize, flow control packets shall be sent with the flow control init operand, 0x1.

C7-56: When in the PortStates LinkArm or LinkActive, flow control packets shall be sent with the normal flow control operand, 0x0.

C7-57: All other values of the Op field are reserved for operations that may be defined by IBA in the future. Any packet received with a reserved value shall be discarded.

7.9.4.1.2 FLOW CONTROL TOTAL BLOCKS SENT (FCTBS) - 12 BITS

The FCTBS (Flow Control Total Blocks Sent) field is generated by the transmitter side logic. The calculation for the value of FCTBS is described later.

7.9.4.1.3 FLOW CONTROL CREDIT LIMIT (FCCL) -12 BITS

The FCCL (Flow Control Credit Limit) field is generated by the receiver side logic. The calculation for the value of FCCL is described later.

7.9.4.1.4 VIRTUAL LANE (VL) - 4 BITS

VL (Virtual Lane) is set to the virtual lane to which the FCTBS and FCCL fields apply.

7.9.4.1.5 LINK PACKET CYCLIC REDUNDANCY CHECK (LPCRC) - 16 BITS

LPCRC (Link Packet Cyclic Redundancy Check) field is a 16-bit CRC that covers the first four bytes of the flow control packet. See Section 7.8.3, “Link Packet CRC (LPCRC) - 2 Bytes,” on page 171.

7.9.4.2 CALCULATION OF FCTBS

C7-58: Upon transmission of a flow control packet, FCTBS shall be set to the total blocks transmitted in the virtual lane since link initialization.

C7-59: When in the PortState initialize, FCTBS shall be set to zero.

7.9.4.3 CALCULATION OF FCCL

The FCCL calculation is based on a 12-bit Adjusted Blocks Received (ABR) counter maintained for each virtual lane at the receiver.

C7-60: The ABR counter shall be set to zero when in the PortState initialize.

C7-61: Upon receipt of each flow control packet, the ABR shall be set to the value of the FCTBS field.

C7-62: Upon receipt of each data packet, the ABR shall be increased by the blocks received, modulo 4096, except that the ABR shall not be increased for received packets that are discarded due to lack of receive capacity in the receiver.

C7-63: The FCCL field shall be set as specified in Section 7.9.4.3, "Calculation of FCCL," on page 184.

Upon transmission of a flow control packet, FCCL shall be set to one of the following:

- If the current buffer state of the receiver would permit reception of 2048 or more blocks from all combinations of valid IBA packets without discard, then the FCCL shall be set to ABR plus 2048 modulo 4096.
- Otherwise the FCCL shall be set to ABR plus the number of blocks the receiver is capable of receiving from all combinations of valid IBA packets without discard modulo 4096.

The number of blocks the receiver is capable of receiving means the number that the receiver can guarantee to receive without buffer overflow regardless of the sizes of the packets that arrive. If, for example, a receiver is capable of receiving more data when large packets arrive than for small packets, the receiver must use the smaller capacity to calculating FCCL.

This specification does not preclude the reconfiguration of receive buffers while the link is active. Such reconfiguration may result in changes of the FCCL value, including the possibility of reduction of available credit. Also, link errors may cause discrepancies between ABR at the receiver and FCTBS at the transmitter. When this has happened, the next flow control update to the receiver will correct the value of ABR and may result in changes of FCCL which reduce or increase credit. When FCCL is updated, the credit calculation for outgoing data packets should use the new value. Packets that are currently being transmitted or queued may be sent based on the previous FCCL value.

7.9.4.4 TRANSMISSION OF PACKETS

If a data packet is available for transmission:

- Let CR represent the total blocks sent since link initialization plus the number of blocks in the data packet to be transmitted, all modulo 4096.
- Let CL represent the last FCCL received in a flow control packet.

If (CL-CR) modulo 4096 $\leq$ 2048, then the data packet may be transmitted. If the condition is not true, then the data packet may not be transmitted until the condition becomes true. Flow control packet transmission is not subject to this restriction nor are any packets on virtual lane 15.

C7-64: A non-VL15 data packet may only be sent when there is sufficient credit as determined by the calculation in Section 7.9.4.4, "Transmission of Packets," on page 185.

C7-65: VL15 packets shall not be subject to flow control.

7.10 IBA AND RAW PACKET MULTICAST

7.10.1 OVERVIEW

Multicast is a one-to-many communication paradigm designed to improve the efficiency of communication between a set of end nodes. Figure 61 illustrates an example unreliable multicast IBA operation:

- A packet with PSN = 1129 is received on an IBA routing element (switch or router) port.
- Switches extract the multicast DLID from the LRH to determine if it corresponds to a multicast group. An implementation may maintain this data as part of its internal route table, e.g. a bit-mask which corresponds to the output ports this packet should be forwarded.
- Routers extract the GID from the GRH for IBA multicast or, for raw packet support, examine the IPv6 header or Ethertype within the RWH to determine if the packet corresponds to a multicast group. It uses this information to forward the packet to the next hop(s) to the destination(s).
- Switches or routers replicate the packet (implementation dependent) and forward the packet onto the output port(s).

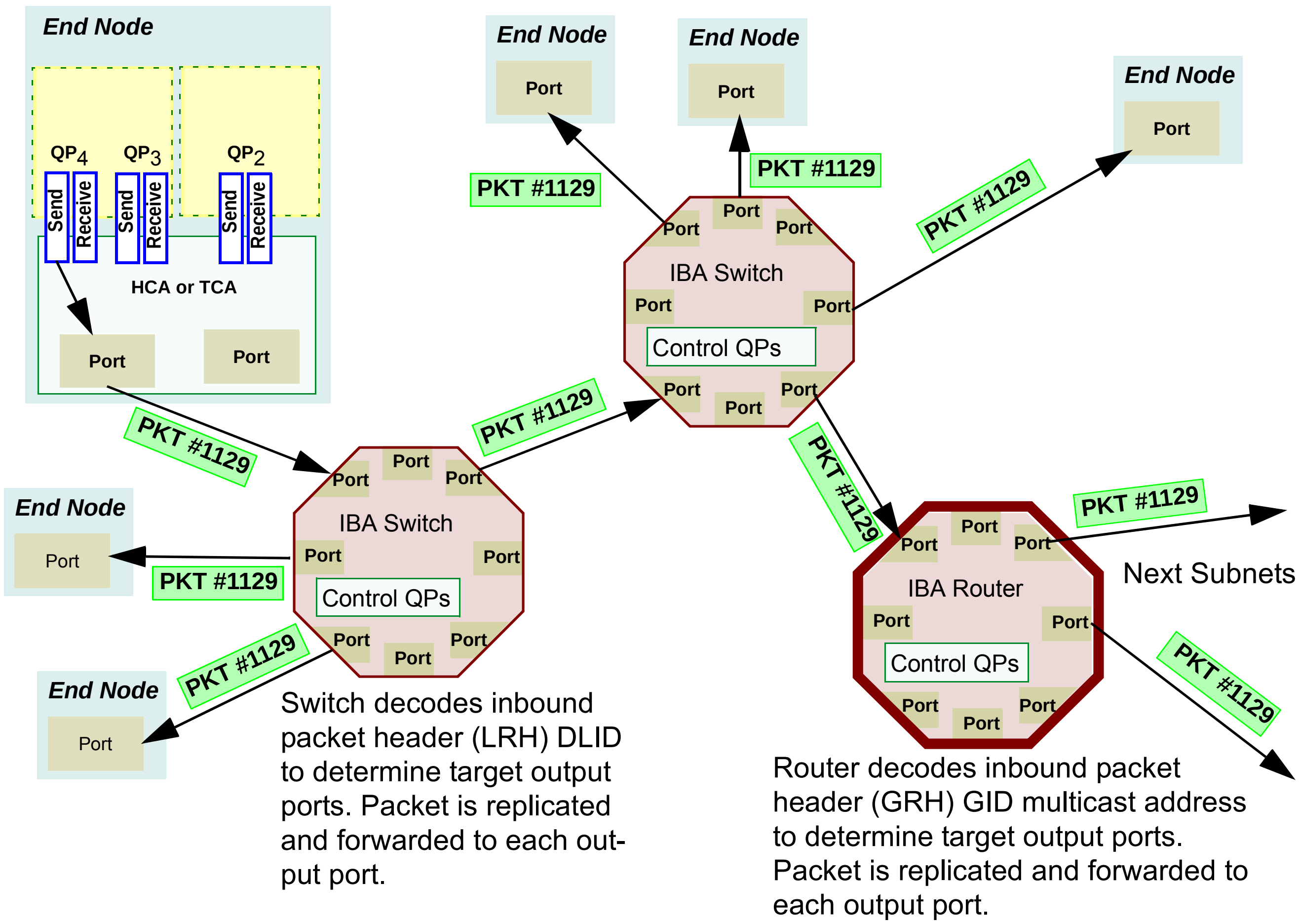


Figure 61 Example IBA Unreliable Multicast Operation

7.10.2 IBA UNRELIABLE MULTICAST OPERATIONAL RULES

o7-13: IBA unreliable multicast is an optional capability. When implemented, it shall function based on the operational rules in Section 7.10.2, "IBA Unreliable Multicast Operational Rules," on page 187.

1) Multicast capability discovery, route table modification, status, and control shall be administered by an IBA management entity. Refer to 15.2.5.8 MulticastForwardingRecord on page 714 and 14.2.5.12 MulticastForwardingTable on page 678".

2) Within the network, packets are replicated within IBA switches and routers and forwarded to the corresponding output ports.

3) Packets are not reliable with respect to acknowledgment generation nor delivery guarantees.

4) Switches and routers may vary in their ability to support multicast packets and thus may have implementation-specific scheduling, resource management, and congestion policies which are outside the scope of IBA.

5) Application multicast packets may be transmitted on VLs as assigned via the SL to VL mapping table by the subnet manager. The use of VL 15 for multicast is prohibited.

6) Application multicast packet headers may contain any SL as provided or derived from values provided by the subnet manager.

7) Applications targeting a multicast group use a multicast group GID - each CA or router port participating in a multicast group shall be assigned the corresponding multicast group GID.

8) Each CA, switch or router that supports multicast may participate in zero, one, or many multicast groups.

9) Multicast groups may span multiple subnets - a multicast capable router is required to forward packets to the next hop to the destination.

10) Multicast packets may be generated by either a CA or a router.

11) Multicast group membership is opaque to the participating end nodes, i.e. it is impossible to know which end nodes are participating within a multicast group and whether all participating end nodes within a multicast group reside within a local or remote subnet. Therefore, all IBA multicast packets shall contain a GRH with the destination multicast GID defined per the IBA addressing rules.

12) The SGID within the GRH shall be set to the source port which initially injected the packet into the network.

13) Messages shall be limited to single-packet messages. The maximum message size is set during the multicast group's creation. The group creator sets the Path MTU (PMTU) for the multicast group. A CA / router will query the SM for the PMTU during multicast group join operation.

    - If an end node attempts to join a multicast group and is unable to accept the current PMTU, the join operation must fail.

14) For each multicast group a CA port is participating in, the CA port shall associate at least one locally managed QP.

    - If a source port is also a destination port within the destination multicast group, the source shall internally replicate the packet within the channel interface to the associated local QPs.

- If the destination end node contains multiple locally managed QPs participating in a multicast group, the destination end node is responsible for internally replicating the packet within the channel interface and delivering a copy to each QP.

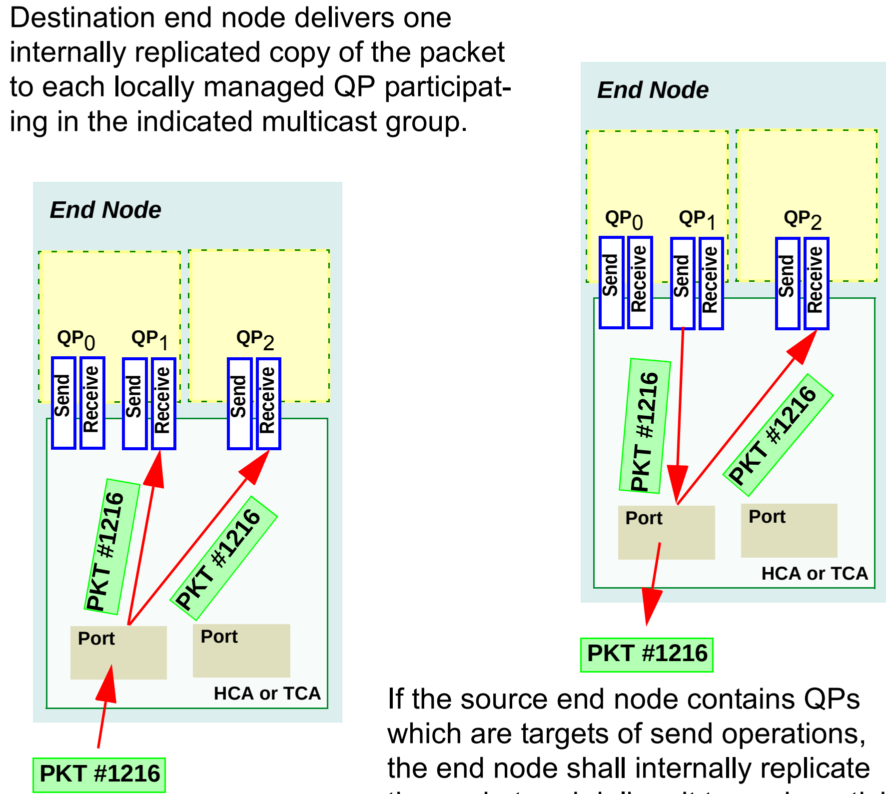


Figure 62 Packet Delivery within an end node

15) Unreliable multicast shall use the unreliable datagram transport service. Refer to the unreliable datagram transport services section for operational rules, constraints, verification, and error handling.

16) A source end node shall set the destination QP within the packet header to 0xFFFFFF.

7.10.3 Raw Packet Multicast

Raw packets may be multicast using the same basic principles as unreliable multicast IBA packets.

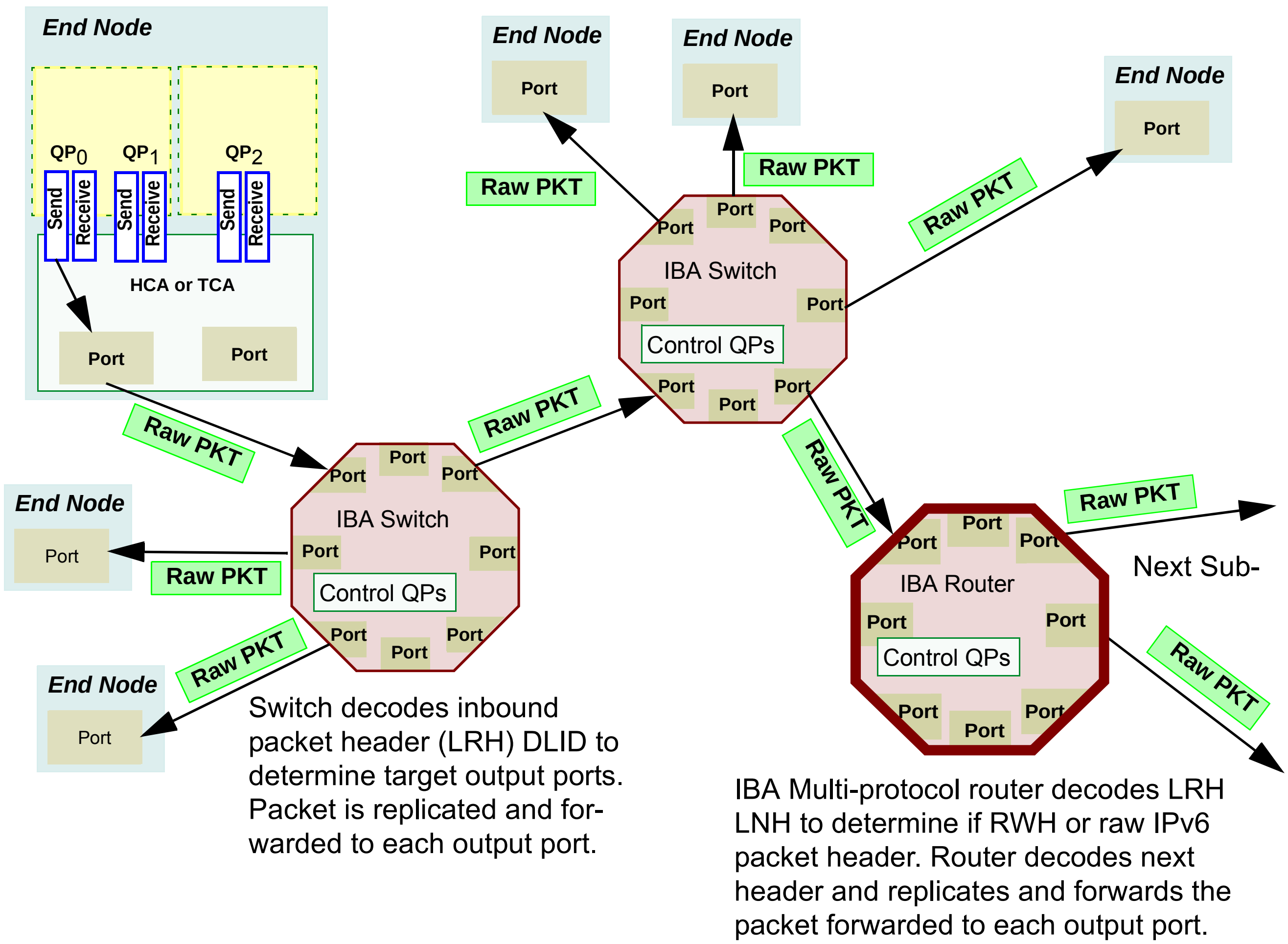


Figure 63 Example Raw Packet Multicast Operation

7.10.3.1 Raw Multicast Operational Rules

o7-14: Raw packet unreliable multicast is an optional capability. When implemented, it shall function based on the operational rules in Section 7.10.3, “Raw Packet Multicast,” on page 190.

1) Raw packet multicast is optional functionality defined within IBA.

2) Raw multicast capability discovery, route table modification, status, and control shall be administered by an IBA management entity.

3) Within the network, packets are replicated within IBA switches and forwarded to the corresponding output ports.
   - Switches extract the multicast DLID from the LRH to determine the corresponding output ports.
4) Routing elements may vary in their ability to support multicast packets and thus may have implementation-specific scheduling, resource management, and congestion / drop policies which are outside the scope of this architecture.
5) Raw multicast packets may be transmitted on any VL except VL 15.
6) Raw multicast packets may be transmitted using any valid SL.
7) IPv6 applications target a multicast group using an IPv6 multicast address. All other protocols use protocol specific addressing and resolution.
8) Each CA or router which supports multicast may participate in zero, one, or many multicast groups.
9) Raw multicast groups may span multiple subnets - a multicast capable router is required to forward packets to the next hop to the destination.
10) Raw multicast packets may be generated by either a CA or a router.
11) Messages shall be limited to single-packet messages. The maximum message size is a function of the PMTU between the source and destination end nodes. Raw protocol management will interact with IBA management entity to determine the maximum PMTU allowed. Raw multicast operations are not required to fail if the PMTU is too small - error recovery is the responsibility of the raw multicast group management protocol.
12) Raw packet support requires a minimum of one locally managed QP. An implementation may provide additional QPs based on implementation-specific policies. As such, implementations are responsible for local raw packet replication and delivery.
    - If a source port is also a destination port within the destination multicast group, the source shall internally replicate the packet within the channel interface to the associated local application targets.
    - If the destination end node contains multiple participating application targets within a raw multicast group, the destination end node is responsible for internally replicating the packet within the channel interface and delivering a copy to each target.
13) Raw packet multicast shall use the IBA raw packet header formats and semantics.

7.10.4 Group Management

IBA Release 1.0 does not fully define the multicast group management protocol used to implement join and leave operations. However, the management section does contain the management interface and associated MADs to implement a multicast group protocol. Refer to 15.2.5.18 MCGroupRecord on page 723".

7.11 Subnet Multipathing

7.11.1 Multipathing Requirements on end node

Each CA and router port is initialized with a LID plus an LMC (LID Mask Control) by the subnet manager. The value of LMC indicates the number of low order bits of the LID to mask when checking a received DLID against the port's DLID. LMC may take values from 0 to 7. Therefore, a port may be identified by 1 to 128 unicast LIDs.

C7-66: When a link layer of a CA or router port checks that a unicast DLID in a received packet is a valid DLID for that port, it shall mask the number of low order bits indicated by the LMC before comparing the DLID to its assigned LID.

The subnet manager may program alternate paths through the subnet for these various LIDs. The selection of which LID to use in the SLID and DLID of transmitted packets is covered in the Transport chapter.

7.12 Error detection and handling

7.12.1 Error Detection

The following classes of errors are detectable by the link layer:

- Single packet receive errors
  - Local physical errors - errors indicative of bit errors on the attached physical link. Failures of ICRC, LPCRC and VCRC checks in the packet check state machines and entry to the bad packet state of the packet receiver state machine belong to this class.
  - Remote physical errors - errors indicative of bit errors on a link other than the attached physical link. Entry to the marked bad packet state of the packet receiver state machine belongs to this class.
  - Malformed packet errors - errors indicative of packets transmitted with inconsistent content. The packet was possibly bad at the source. It is also possible that the error was inserted by a switch. Programming errors of switch or port configuration by the SM

may also create errors in this category. LVer error, Length error, op_code error, VL error, and GRH_VL15 error belong to this class. These are all errors from the packet check state machines.

- Switch routing errors - errors indicative of an error in switch routing. DLID errors are in this class.
- Buffer overrun - error indicative of an error in the state of the flow control machine in the link layer at the other end of the physical link. One cause of such an error can be an earlier packet with a physical error if buffers are not immediately reclaimed from bad packets.

C7-67: An error in a received packet shall be classified as specified in Section 7.12.1, "Error Detection," on page 192.

C7-68: When error counters for the single packet receive errors are implemented and one or more errors are detected in a received packet, then the counter associated with the error with the highest precedence as defined by Section 7.3, "Packet Receiver States," on page 146, Section 7.4, "Data Packet Check," on page 148, and Section 7.5, "Link Packet Check," on page 151 shall increment and none of the other single packet error counters shall increment.

- Receiver errors
  - Local link integrity - excessively frequent local physical errors. This error is caused by a marginal link. A more severe physical problem will be detected at the physical layer based on high frequency of code violations. Detection of local link integrity errors is based on a count of local physical errors. The count starts at zero and shall be incremented for each packet received with a local physical error. If the current count is above zero, the counter shall be decremented once for each packet received without a local physical error. When it exceeds local_phy_errors threshold, the local link integrity error shall be detected.
  - Excessive buffer overruns - buffer overrun errors persisting over multiple flow control update times. This error shall be detected when overrun_errors_threshold consecutive flow control update periods occur with at least one overrun error in each period.

C7-69: Each port shall implement detection of local link integrity and excessive buffer overrun errors as specified in Section 7.12.1, "Error Detection," on page 192.

- Transmitter errors
  - Flow control update - errors indicative of a failure of the flow control machine at the other end of the link. For each VL active in the current port configuration, except VL 15 there shall be a watchdog timer monitoring the arrival of flow control updates. If the tim-

er expires without receiving an update, a flow control update error has occurred. The period of the watchdog timer shall be 400,000 +3%/-51% symbol times. This timer shall only run when PortState = Arm or Active. When PortState = ActiveD, this timer shall be reset. When PortState = Initialize or when a flow control packet is received, the timer shall be reset.

C7-70: Each port shall implement detection of flow control update errors as specified in Section 7.12.1, "Error Detection," on page 192.

7.12.2 ERROR RECOVERY PROCEDURES

The response to any single packet receive error is to discard the packet. No further recovery is necessary at the link layer. For some errors, the data packet check state machine (Section 7.4, "Data Packet Check," on page 148) allows a switch to forward a packet with an error marking it as bad by appending a bad VCRC value and the EBP delimiter as an alternative to dropping the packet.

Local link integrity, excessive buffer overrun, and flow control update errors all indicate errors that may be fixed by retraining or may be due to a hard fault.

C7-71: Upon detecting local link integrity, excessive buffer overrun or flow control update errors, the link shall initiate retraining by asserting L_init_train (refer to 6.3.1.2 L_Init_Train - Link Initiate Retraining on page 138).

7.12.3 ERROR NOTIFICATION

Single packet receive error classes increment error counters as specified in management (Refer to 16.1.4 Optional Attributes on page 767). Note that at most one link layer error is detected per packet so each packet increments one and only one of these counters.

Local link integrity, excessive buffer overrun, and flow control update are counted and may produce a trap as specified in management.

CHAPTER 8: NETWORK LAYER

8.1 OVERVIEW

This chapter describes the network layer within IBA. Within the IBA layered architecture, this layer is responsible for routing packets between IBA subnets. This includes unicast and multicast operations. This chapter specifies routing between IBA subnets - it does not specify multi-protocol routing, i.e. routing IBA over non-IBA fabric types, nor does it specify how raw packets are routed between IBA subnets.

This chapter, with the exception of section 8.4 Global Route Header Usage on page 199, is informational in nature. As such, it does not specify IBA requirements; refer to Chapter 19: Routers on page 862 for requirements of IBA routers. Packet forwarding within an IBA subnet is done at the link layer by IBA switches; refer to Chapter 18: Switches on page 845 for requirements of IBA switches.

8.2 PACKET ROUTING

8.2.1 OVERVIEW

IBA supports a two-layer topological division. The lower layer is referred to as an IBA subnet. Packets are forwarded throughout the subnet utilizing IBA switches (the process of forwarding a packet from one link to another within a subnet is referred to as switching). The path that a packet takes through this layer is uniquely defined by its point of injection into the fabric, identified in the packet by the SLID field in the LRH, and the DLID and SL fields in its LRH.

At the higher layer, subnets are interconnected using routers (the process of forwarding a packet from one subnet to another is referred to as routing). Routing may be accomplished utilizing routers conforming to the IBA specification, and may also be accomplished using routers conforming to other specifications (e.g. utilizing the Internet Protocol (IP) suite of specifications). The series of subnets through which a packet passes is not defined by IBA; however, several fields are provided in the Global Route Header to enable routers to make this decision. These fields include SGID, DGID, TClass and FlowLabel. Additionally, a router might use fields from other headers, e.g. the SL field in the LRH to determine a mapping to TClass. Regardless of the mechanism used to in forwarding decisions, IBA requires that the path be symmetric with respect to SGID and DGID. This means that if a valid path exists from an SGID to a DGID, then IBA requires that a valid path also exist swapping the values of DGID and SGID.

The requirements of IBA routers are specified in Chapter 19: Routers on page 862. Interconnection of IBA subnets utilizing IBA routers is intended to preserve IBA intra-subnet behavior across subnets.

Use of other routing technologies is beyond the scope of IBA; however, the architecture is intentionally crafted to enable this capability, especially utilizing IP version 6 as specified by IETF RFC 2460 and other associated IETF RFCs.

A global IBA fabric consists of one IBA subnet or multiple IBA subnets interconnected via routers. As described above, this global fabric may also include non-IBA interconnects between IBA subnets, as well as gateways to non-IBA fabrics.

8.2.2 GLOBAL FABRIC CHARACTERISTICS

This section describes the characteristics of a global fabric interconnected exclusively with IBA routers. While beyond the scope of IBA, global fabrics interconnected with non-IBA technology may also exhibit some or all of these characteristics.

8.2.2.1 INHERITANCE OF SUBNET REQUIREMENTS

All the packet delivery characteristics of a subnet are inherited by the global fabric, except for virtual lane 15 subnet management packets (since subnet management occurs at the subnet level, these packets do not transit routers).

8.2.2.2 PACKET ERRORS AND ERROR DETECTION

IBA specifies an invariant CRC that is appended to all IBA packets except raw packets (refer to section 7.8.1 Invariant CRC (ICRC) - 4 Bytes on page 168). This CRC covers all of the IBA packet fields that do not require modification to effect IBA routing. End-to-end data integrity assurance is provided by retaining this CRC unmodified as the packet transits the global fabric.

8.2.2.3 SERVICE LEVELS

Service levels and virtual lanes are supported throughout the global fabric. This is accomplished by mapping service level to traffic class in the GRH, and vice versa. The mapping function itself, as is the interpretation of service level, is beyond the scope of IBA.

8.2.3 SUPPORT FOR MULTIPLE GLOBAL PATHS

The information required to route a packet within a subnet and between subnets is contained in the packet's local route header and global route headers, respectively. Unlike many network protocols, IBA does not require a packet to contain a global route header unless the packet is either destined for a device that is not on the same subnet or the packet is a multicast packet. However, any packet except subnet management packets may contain a global route header (subnet management packets are defined in 14.2.1 Datagram Formats and Use on page 642.)

The identification and utilization of multiple paths between two channel adapters on different subnets is hierarchical and involves similar but independent mechanisms within subnets and across subnets.

Within subnets, multiple paths between two channel adapters are identified by multiple LIDs. That is, a port may effectively be assigned multiple LIDs using a LID/LMC combination Chapter 4: Addressing on page 115. The source channel adapter indicates a path via its selection of one of the LIDs assigned to the destination port.

Likewise, channel adapters have the option to support the assignment of multiple GIDs. In the case of global routing across subnets, the LID indicates which of the valid paths is to be used within the subnet (i.e. switch forwarding) and the GID indicates which of the valid paths is to be used between subnets (i.e. router forwarding).

As a packet transits a subnet, its SLID and DLID fields remain unchanged. As a packet transits between subnets (i.e. through a router), the router updates the SLID to that of its own LID and the DLID to the LID of the next router or final destination, as appropriate.

Note that for global routing, this provides two degrees of freedom for a source channel adapter to select a path through the fabric. Selection of the LID determines the route through the subnet to the first router. Selection of the GID determines the route taken after reaching the first router. Each router along the path may choose the path through a subnet to the next router (or final destination) via its selection of the LID for the next router (or final destination). Furthermore, since the DLID may contain LMC bits of multipath data, the router may use the DLID as part of its route determination algorithm.

The decision process that routers use for forwarding packets is not specified by IBA; however, routers may rely on various combinations of Destination GID, Source GID, SL, TClass, and FlowLabel fields, among other factors, to determine the forwarding path and flows that must exhibit in-order delivery. Channel Adapters and/or ingress routers may label flows of packets that are expected to be delivered in order with the same FlowLabel in the global route header. While IBA routers utilize LIDs and GIDs to determine paths, the FlowLabel may be used by non-IBA routers to determine paths.

8.2.4 GLOBAL MULTICAST

IBA supports an unreliable multicast mechanism. A detailed description of this mechanism may be found in section 7.10 IBA and Raw Packet Multicast on page 186. Implementation of this mechanism is optional in IBA devices (including switches and routers). Multicast packets within a given multicast group, i.e. multicast packets that share a common multicast GID, may be sourced by a single device or by multiple devices. Since routers are not fully specified by IBA, routers may vary in their ability to support multicast packets and may have implementation specific.

8.3 GLOBAL ROUTE HEADER

Figure 64 on page 198 illustrates the format of the Global Route Header that is used for inter-subnet routing.

Figure 64 Global Route Header (GRH)

bits / bytes	31-24		23-16	15-8	7-0
0-3	IPVer	TClass		FlowLabel	
4-7	PayLen			NxtHdr	HopLmt
8-11	SGID[127-96]				
12-15	SGID[95-64]				
16-19	SGID[63-32]				
20-23	SGID[31-0]				
24-27	DGID[127-96]				
28-31	DGID[95-64]				
32-35	DGID[63-32]				
36-39	DGID[31-0]				

Global route headers are not required in all packets (see section 8.4.1 Global Route Header Generation on page 199 for details). The presence of a Global Route Header is indicated in the Local Route Header as specified in 7.7.5 Link Next Header (LNH) - 2 bits on page 167. The following subparagraphs describe the fields in the GRH:

8.3.1 IP VERSION (IPVER) - 4 BITS

Indicates the version of the GRH; always set to 6.

8.3.2 TRAFFIC CLASS (TCLASS) - 8 BITS

This field is used to communicate service level end-to-end, i.e. across subnets. The mapping of specific traffic class to specific TClass values is not specified by IBA and may vary by implementation.

8.3.3 Flow Label (FlowLabel) - 20 bits

This field may be used to identify a sequence of packets that must be delivered in order.

8.3.4 Payload Length (PayLen) - 16 bits

For an IBA packet this field specifies the number of bytes starting from the first byte after the GRH up to and including the last byte of the ICRC. For a raw IPv6 datagram this field specifies the number of bytes starting from the first byte after the GRH up to but not including either the VCRC or any padding to achieve a multiple of a 4 byte packet length.

8.3.5 Next Header (NxtHdr) - 8 bits

This field indicates what header, if any, follows the global route header.

8.3.6 Hop Limit (HopLmt) - 8 bits

This field indicates the number of hops (i.e. the number of routers transited) that the packet is permitted to take prior to being discarded. This ensures that a packet will not loop indefinitely between routers should a routing loop occur. Setting this value to 0 or 1 will ensure that the packet will not be forwarded beyond the local subnet.

8.3.7 Source Global Identifier (SGID) - 128 bits

This field identifies the port that injected the packet into the global fabric. Additional information on the format and use of GID's may be found in Chapter 4: Addressing on page 115.

8.3.8 Destination Global Identifier (DGID) - 128 bits

This field identifies the final destination port of the packet, or to the multicast group that represents the set of ports to which the packet is to be delivered. Additional information on the format and use of GID's may be found in Chapter 4: Addressing on page 115.

8.4 Global Route Header Usage

The following subsections describe the usage of the global route header:

8.4.1 Global Route Header Generation

C8-1: A channel adapter initiating a packet shall include a global route header if any of the following conditions apply:

- The packet is a multicast packet.
- The final destination of the packet is a port of a device that is not on the same subnet as the port that initially injects the packet into the fabric and both the injecting and receiving ports are connected to IBA subnets.

o8-1: A channel adapter, switch, or router initiating a packet may include a global route header in any packet except for SMPs.

If a global route header is included, the fields are loaded by the initiating channel adapter, switch, or router as follows:

C8-2: IPVer: If a global route header is included in a packet, this field shall be set to 6.

C8-3: TClass: If a global route header is included in a packet, this field shall either be set to zero or to an appropriate TClass value by the injecting channel adapter. Each router maps TClass to a SL appropriate for the subnet on which it will inject the packet. This mapping function is not specified by IBA.

FlowLabel: The use of this field is not required by IBA.

C8-4: If a global route header is included in a packet, and FlowLabel is not used, it shall be set to zero.

C8-5: If a global route header is included in a packet and FlowLabel is used, all packets that must be delivered in order with respect to each other shall be identified by a constant, non-zero value inserted in the FlowLabel field.

This implies that if a given QP uses a non-zero flow label, it must use the same flow label on all packets emitted from that QP that are destined for a given remote QP. Different QPs transmitting to a given destination may use the same or different flow labels. Flow labels may be shared among QPs.

C8-6: PayLen: If a global route header is included in an IBA packet, this field shall be loaded with the length of the packet, in bytes, starting from the first byte after the global route header up to and including the last byte of the ICRC.

NxtHdr: The use of this field varies depending on whether the packet is a raw or non-raw packet.

C8-7: For non-raw IBA packets that include a GRH, the NxtHdr field shall contain *0x1B*.

C8-8: For raw packets that include a IPv6 header, the contents of NxtHdr shall be set to the identifier for the next header as defined in IETF RFC 1700 et. seq.

C8-9: HopLmt: If a global route header is included in a packet, this field shall be set to the number of hops (i.e. the number of routers that may be transited) that the packet is permitted to take prior to being discarded.

C8-10: SGID: If a global route header is included in a packet, this field shall be set to one of the GID's assigned to the port that will inject the packet into the fabric.

C8-11: DGID: If a global route header is included in a packet, this field shall be set to one of the GID's assigned to the port that is the final destination of this packet, or to the multicast GID that represents the set of ports to which the packet is to be delivered.

8.4.2 GLOBAL ROUTE HEADER MODIFICATION

This section describes the modifications that may and must be made to the global route header by IBA routers when forwarding packets between subnets. Note that modification of these fields implies updating the packet's variant CRC defined in 7.8.2 Variant CRC (VCRC) - 2 Bytes on page 170. These changes do not affect the packet's invariant CRC defined in 7.8.1 Invariant CRC (ICRC) - 4 Bytes on page 168.

C8-12: IPVer: This field shall not be changed by IBA routers.

TClass: This field is used to communicate service level end-to-end, i.e. across subnets. Routers utilize this field to determine an appropriate SL for forwarding on the next subnet. This mapping function is not specified by IBA.

C8-13: The TClass field, if non-zero, shall not be modified by IBA routers.

The use of TClass by routers when it contains zero is not defined by IBA.

FlowLabel: This field may be used to identify a sequence of packets that must be delivered in order. The use of this field is not required by IBA. If not used, it is left unchanged. If used, all packets that must be delivered in order with respect to each other shall be identified by a constant, non-zero value inserted in this field in each packet.

o8-2: The router may change the value of FlowLabel; however, it must use the same flow label for all packets that must be delivered in order, which includes all traffic between any given two QPs.

C8-14: PayLen: IBA routers shall not modify the content of PayLen.

C8-15: NxtHdr: IBA routers shall not modify the content of NxtHdr.

C8-16: HopLmt: IBA routers shall discard packets that contain a value of one or zero in the HopLmt field. Otherwise, IBA routers shall decrement the HopLmt field by one.

C8-17: SGID: IBA routers shall not modify the content of SGID

C8-18: DGID: IBA routers shall not modify the content of DGID.

8.4.3 GLOBAL ROUTE HEADER VERIFICATION

This section describes the verification that must be performed by the network layer at the final destination of the packet. This verification assumes that the packet has passed the verification required of the lower layers to permit the packet to be presented to the network layer.

C8-19: The network layer shall silently discard, with the exception of adjusting any applicable management counters specified elsewhere in this specification, packets that meet any of the following conditions:

- Value of IPVer is not 6.
- The value of DGID does not equal one of the GID values assigned to the port that received the packet.

If none of the above conditions require discard of the packet by the network layer, the network layer presents the packet to the transport layer. Note that the other layers, including the transport layer, may require additional verification of fields within the global route header.

CHAPTER 9: TRANSPORT LAYER

9.1 OVERVIEW

Each IBA packet contains a transport header. The transport header contains the information required by the endnode to complete the specified operation, e.g. delivery of data payload to the appropriate entity within the endnode such as a thread or IO controller. This chapter defines the transport services used by IBA.

The client of an IBA channel adapter communicates with the transport layer by manipulating a “queue pair” (QP) made up of a Send work queue and a Receive work queue. For a host platform, the client of the transport layer is the Verbs software layer. The client posts buffers or commands to these queues and hardware transfers data from or into the buffers. Throughout this chapter, a QP that initiates an operation, i.e. injects a message into the fabric, is referred to as the requester and the QP that receives the message is referred to as the responder.

When a QP is created, it is associated with one of four IBA transport service types or non-IBA protocol encapsulation services. The transport service describes the degree of reliability and to what and how the QP transfers data.

The four IBA transport service types are:

1) Reliable Connection
2) Reliable Datagram
3) Unreliable Datagram
4) Unreliable Connection

The non-IBA protocol encapsulation services are:

1) Raw IPv6 Datagram
2) Raw Ethertype Datagram

Table 256 Channel Adapter Attributes on page 831 lists which of these services are required for Host Channel Adapters and Target Channel Adapters. Table 34 below compares several key attributes of these five transport service types.

Reliable transport services use a combination of sequence numbers and acknowledgment messages (ACK / NAK) to verify packet delivery order, prevent duplicate packets and out-of-sequence packets from being processed, and to detect missing packets. Upon error detection, e.g. a missing packet, the missing packet along with all subsequent packets will be retransmitted by the requestor. IBA does not support selective packet retransmission nor the out-of-order reception of packets.

An IBA operation is defined to include a request message and, for reliable services, its corresponding response. Thus, the request message is generated by a requester, and a response, if one exists, is generated by the responder.

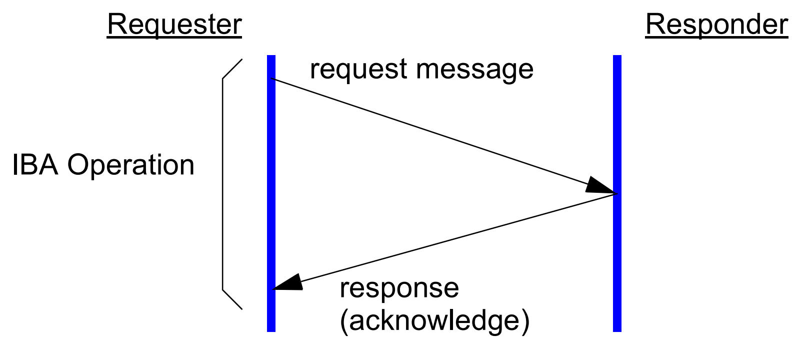


Figure 65 IBA Operation

A request message consists of one or more IBA packets. The packets of a request message are called request packets. A response, except for an RDMA READ Response, consists of exactly one packet. A response is also called an acknowledge. The response packet acknowledges receipt of one or more packets. The response may acknowledge the receipt of packets that comprise anywhere from a portion of a request message to multiple request messages.

Unreliable transport services do not use acknowledgment messages. They do however generate sequence numbers. This allows a responder to detect out-of-sequence or missing packets and to perform local recovery processing. The specifics of any recovery processing for unreliable datagrams are outside the scope of the IBA specification.

Table 34 Comparison of IBA Transport Service Types

<table>
<tr><th colspan="2">Attribute</th><th>Reliable Connection</th><th>Reliable Datagram</th><th>Unreliable Datagram</th><th>Unreliable Connection</th><th>Raw Datagram (both IPv6 & ethertype)</th></tr>
<tr><td colspan="2">Scalability (M processes on N Processor nodes communicating with all processes on all nodes)</td><td>M^2*N QPs required on each processor node, per CA</td><td>M QPs required on each processor node, per CA.</td><td>M QPs required on each processor node, per CA.</td><td>M^2*N QPs required on each processor node, per CA.</td><td>Minimum 1 QP required on each end node, per CA.</td></tr>
<tr><td rowspan="5">Reliability</td><td>Corrupt data detected</td><td colspan="5">Yes</td></tr>
<tr><td>Data delivery guarantee</td><td colspan="2">Data delivered exactly once</td><td colspan="3">No guarantees</td></tr>
<tr><td>Data order guaranteed</td><td>Yes, per connection</td><td>Yes, packets from any one source QP are ordered to multiple destination QPs.</td><td>No</td><td>Unordered and duplicate packets are detected.</td><td>No</td></tr>
<tr><td>Data loss detected</td><td colspan="2">Yes</td><td>No</td><td>Yes</td><td>No</td></tr>
<tr><td>Error recovery</td><td colspan="2">Reliable. Errors are detected at both the requestor and the responder. The requestor can transparently recover from errors (retransmission, alternate path, etc.) without any involvement of the client application. QP processing is halted only if the destination is inoperable or all fabric paths between the channel adapters have failed.</td><td>Unreliable. Packets with some types of errors may not be delivered. Neither source nor destination QPs are informed of dropped packets.</td><td>Unreliable. Packets with errors, including sequence errors, are detected and may be logged by the responder. The requestor is not informed.</td><td>Unreliable. Packets with errors are not delivered. The requestor and responder are not informed of dropped packets.</td></tr>
<tr><td colspan="2">RDMA and ATOMIC Operations</td><td>Yes</td><td>Yes</td><td>No</td><td>Yes: RDMA WRITEs
No: RDMA READs & ATOMICs</td><td>No</td></tr>
<tr><td colspan="2">Bind Memory Window</td><td>Yes</td><td>Yes</td><td>No</td><td>Yes</td><td>No</td></tr>
<tr><td colspan="2">IBA Unreliable Multicast Support</td><td>No</td><td>No</td><td>Yes</td><td>No</td><td>No</td></tr>
<tr><td colspan="2">Raw Multicast</td><td>No</td><td>No</td><td>No</td><td>No</td><td>Yes</td></tr>
<tr><td colspan="2">Message Size</td><td colspan="2">Message size 0 to 2^{31} bytes. Smaller max size may be negotiated by Connection Management. A message may consist of multiple packets.</td><td>Single PMTU packet datagrams - 0 to 4096 bytes.</td><td>Message size 0 to 2^{31} bytes. Smaller max size may be negotiated by Connection Management. A message may consist of multiple packets.</td><td>Single PMTU packet datagrams - 0 to 4096 bytes.</td></tr>
<tr><td colspan="2">Connection Oriented?</td><td>Connected. The client connects the local QP to one and only one remote QP. No other traffic flows over these QPs.</td><td>Connectionless. Appears connectionless to the client - uses one or more End-to-End contexts per CA to provide reliability service.</td><td>Connectionless. No prior connection is needed for communication.</td><td>Connected. The client connects the local QP to one and only one remote QP. No other traffic flows over these QPs.</td><td>Connectionless. No prior connection is needed for communication.</td></tr>
</table>

9.2 BASE TRANSPORT HEADER

Base Transport Header (BTH) contains fields always present for all IBA transport services - it is not present in Raw packets. The presence of BTH is indicated by the Link Next Header (LRH:LNH) field.

C9-1: All IBA transport services shall include a Base Transport Header (e.g. it is not present in Raw packets).

Figure 66 Base Transport Header (BTH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	OpCode	SE M Pad TVer	Partition Key	
4-7	Reserved 8 (masked in ICRC)	Destination QP		
8-11	A Reserved 7	PSN - Packet Sequence Number		

9.2.1 OPERATION CODE (OPCODE)

The OpCode field defines the interpretation of the remaining header and payload bytes. The OpCode list definition is shown in Table 35 OpCode field on page 207.

C9-2: Table 35 shall be used to define the OpCode parameter in the BTH as well as the headers and payload that follow the BTH.

Table 35 OpCode field

Code[7-5]	Code[4-0]	Description	Packet Contents following the Base Transport header[a]
000 Reliable Connection (RC)	00000	SEND First	PayLd
	00001	SEND Middle	PayLd
	00010	SEND Last	PayLd
	00011	SEND Last with Immediate	ImmDt, PayLd
	00100	SEND Only	PayLd
	00101	SEND Only with Immediate	ImmDt, PayLd
	00110	RDMA WRITE First	RETH, PayLd
	00111	RDMA WRITE Middle	PayLd
	01000	RDMA WRITE Last	PayLd
	01001	RDMA WRITE Last with Immediate	ImmDt, PayLd
	01010	RDMA WRITE Only	RETH, PayLd
	01011	RDMA WRITE Only with Immediate	RETH, ImmDt, PayLd
	01100	RDMA READ Request	RETH
	01101	RDMA READ response First	AETH, PayLd
	01110	RDMA READ response Middle	PayLd
	01111	RDMA READ response Last	AETH, PayLd
	10000	RDMA READ response Only	AETH, PayLd
	10001	Acknowledge	AETH
	10010	ATOMIC Acknowledge	AETH, AtomicAckETH
	10011	CmpSwap	AtomicETH
	10100	FetchAdd	AtomicETH
	10101-11111	Reserved	undefined
001 Unreliable Connection (UC)	00000	SEND First	PayLd
	00001	SEND Middle	PayLd
	00010	SEND Last	PayLd
	00011	SEND Last with Immediate	ImmDt, PayLd
	00100	SEND Only	PayLd
	00101	SEND Only with Immediate	ImmDt, PayLd
	00110	RDMA WRITE First	RETH, PayLd
	00111	RDMA WRITE Middle	PayLd
	01000	RDMA WRITE Last	PayLd
	01001	RDMA WRITE Last with Immediate	ImmDt, PayLd
	01010	RDMA WRITE Only	RETH, PayLd
	01011	RDMA WRITE Only with Immediate	RETH, ImmDt, PayLd
	01100-11111	Reserved	undefined

Table 35 OpCode field

Code[7-5]	Code[4-0]	Description	Packet Contents following the Base Transport header[a]
010 Reliable Datagram (RD)	00000	SEND First	RDETH, DETH, PayLd
	00001	SEND Middle	RDETH, DETH, PayLd
	00010	SEND Last	RDETH, DETH, PayLd
	00011	SEND Last with Immediate	RDETH, DETH, ImmDt, PayLd
	00100	SEND Only	RDETH, DETH, PayLd
	00101	SEND Only with Immediate	RDETH, DETH, ImmDt, PayLd
	00110	RDMA WRITE First	RDETH, DETH, RETH, PayLd
	00111	RDMA WRITE Middle	RDETH, DETH, PayLd
	01000	RDMA WRITE Last	RDETH, DETH, PayLd
	01001	RDMA WRITE Last with Immediate	RDETH, DETH, ImmDt, PayLd
	01010	RDMA WRITE Only	RDETH, DETH, RETH, PayLd
	01011	RDMA WRITE Only with Immediate	RDETH, DETH, RETH, ImmDt, PayLd
	01100	RDMA READ Request	RDETH, DETH, RETH
	01101	RDMA READ response First	RDETH, AETH, PayLd
	01110	RDMA READ response Middle	RDETH, PayLd
	01111	RDMA READ response Last	RDETH, AETH, PayLd
	10000	RDMA READ response Only	RDETH, AETH, PayLd
	10001	Acknowledge	RDETH, AETH
	10010	ATOMIC Acknowledge	RDETH, AETH, AtomicAckETH
	10011	CmpSwap	RDETH, DETH, AtomicETH
	10100	FetchAdd	RDETH, DETH, AtomicETH
	10101	RESYNC	RDETH, DETH
	10110-11111	Reserved	undefined
011 Unreliable Datagram (UD)	00000-00011	Reserved	undefined
	00100	SEND only	DETH, PayLd
	00101	SEND only with Immediate	DETH, ImmDt, PayLd
	00110-11111	Reserved	undefined
100 - 101	00000-11111	Reserved	undefined
110 - 111	00000-11111	Manufacturer Specific OpCodes	undefined

a. All OpCodes have the ICRC and VCRC attached.

9.2.2 RESERVED TRANSPORT FUNCTION OPCODES

For future expansion of its transport layer, IBA provides Reserved and Manufacturer Defined BTH OpCodes. Two blocks of undefined OpCodes are specified: one for future revisions of the IBA and one block for manufacturer specific functions. Manufacturer Defined opcodes should not be used between devices until the devices are clearly identified as supporting those opcodes.

9.2.3 SOLICITED EVENT (SE) - 1 BIT

The requester sets this bit to 1 to indicate that the responder shall invoke the CQ event handler. Additional operational guidelines:

- A CQ event handler must be configured for the target CQ.
- The SE bit should only be set in the last or only packet of a SEND, SEND with Immediate, or RDMA WRITE with Immediate.
- CQ event handler is invoked after a completion event is written to the CQ.

SE bit is not considered a part of packet header validation, i.e. receipt of a packet with this bit set that does not meet the invocation requirements will not result in a NAK being generated.

C9-3: For an HCA, if an inbound request packet has the Solicited Event bit in the BTH to 1 and the additional SE operational guidelines are valid, it shall invoke the CQ event handler.

o9-1: For a TCA supporting Solicited Events, if an inbound request packet has the Solicited Event bit in the BTH to 1 and the additional SE operational guidelines are valid, it shall invoke the CQ event handler.

C9-4: The responder shall not consider the SE bit in the BTH part of the packet header validation.

9.2.4 MIGREQ (M) - 1 BIT

Used to communicate migration state. If set to one, indicates the connection or EE context has been migrated; if set to zero, it means there is no change in the current migration state. See Automatic Path Migration within the Chapter 17: Channel Adapters on page 822.

9.2.5 PAD COUNT (PADCNT) - 2 BITS

Packet payloads are sent as a multiple of 4-byte quantities. Pad count indicates the number of pad bytes - 0 to 3 - that are appended to the packet payload. Pads are used to “stretch” the payload (payloads may be zero or more bytes in length) to be a multiple of 4 bytes.

9.2.6 TRANSPORT HEADER VERSION (TVER) - 4 BITS

Specifies the version of the IBA Transport used for this packet. This version applies to all of the transport fields including the BTH, extended header and the invariant CRC - this field is set to 0x0. If a receiver does not support the Transport Version specified then the packet is discarded.

C9-5: Requesters and responders using IBA transports shall generate IBA transport packets with BTH:TVer = 0x0.

9.2.7 PARTITION KEY (P_KEY) - 16 BITS

P_Key identifies the partition that the destination QP (RC, UC, UD) or EE Context (RD) is a member.

9.2.8 DESTINATION QP (DESTQP) - 24 BITS

This field specifies the destination queue pair (QP) identifier.

9.2.9 RESERVE 8 (RESV8) - 8 BITS

Reserved (variant) - 8 bits. Transmitted as 0, ignored on receive. This field is not included in the invariant CRC.

C9-6: When generating a packet, the sender shall set the Resv8 field to zero. The receiver shall ignore this field.

9.2.10 ACKREQ (A) - 1 BIT

Requests responder to schedule an acknowledgment on the associated QP.

9.2.11 RESERVE 7 (RESV7) - 7 BITS

Transmitted as 0, ignored on receive. This field is included in the invariant CRC.

C9-7: When generating a packet, the sender shall set the Resv7 field to zero. The receiver shall ignore this field.

9.2.12 PACKET SEQUENCE NUMBER (PSN) - 24 BITS

This field is used to identify the position of a packet within a sequence of packets. All IBA requesters shall generate a monotonically increasing (modulo 2^{24}) PSN when originating a packet. Depending upon the transport service type and / or implementation requirements, a responder may validate the PSN to detect missing packets.

9.3 EXTENDED TRANSPORT HEADERS

9.3.1 RELIABLE DATAGRAM EXTENDED TRANSPORT HEADER (RDETH) - 4 BYTES

Reliable Datagram Extended Transport Header (RDETH) contains the End-to-End Context identifier.

Figure 67 Reliable Datagram Extended Transport Header (RDETH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Reserve	EE-Context		

9.3.1.1 RESERVE - 8 BITS

o9-2: If a CA implements Reliable Datagram functionality, then when generating a packet, the sender shall set this field to 0x0. The receiver shall ignore this field.

9.3.1.2 END-TO-END (EE) CONTEXT - 24 BITS

This field indicates the End-to-End (EE) Context used for this packet. EE context is a unique endnode identifier used to multiplex / demultiplex reliable datagram packets between any two end nodes. The EE-Context provides a context for reliable transfer state similar to that used for reliable connection.

9.3.2 DATAGRAM EXTENDED TRANSPORT HEADER (DETH) - 8 BYTES

Datagram Extended Transport Header (DETH) contains the additional transport fields for reliable and unreliable datagram service.

Figure 68 Datagram Extended Transport Header (DETH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Queue Key			
4-7	Reserve	Source QP		

9.3.2.1 Q_KEY - 32 BITS

This field is required to authorize access to the destination queue. The responder compares this field with the destination's QP Q_Key.

9.3.2.2 RESERVE - 8 BITS

C9-8: When generating a packet, the sender shall set this field to 0x0. The receiver shall ignore this field.

9.3.2.3 SOURCE QP (SRCQP) - 24 BITS

This field specifies the source queue pair (QP) identifier. This is used as the destination QP for response packets.

9.3.3 RDMA Extended Transport Header (RETH) - 16 Bytes

RDMA Extended Transport Header (RETH) contains the additional transport fields for RDMA operations.

Figure 69 RDMA Extended Transport Header (RETH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Virtual Address (63-32)			
4-7	Virtual Address (31-0)			
8-11	R_Key			
12-15	DMA Length			

9.3.3.1 Virtual Address (VA) - 64 bits

Start address of buffer. RDMA VA may start on any byte boundary.

9.3.3.2 R_Key - 32 bits

R_Keys have the following properties:

- A R_Key acts as a protection key to access the specified memory address and range for a given operation, i.e. it is a protection mechanism to insure proper access to the target memory. The responder correlates the R_Key to the local protection mechanisms to validate the requester's access rights.
- A R_Key must be exported to the requester - this process (also includes the export of the starting virtual address and memory size, i.e. length) is outside the scope of this section.
- Access rights are granted for any combination of RDMA READ, RDMA WRITE, and ATOMICs - including none and all.
- Each Memory Region or Window has a single valid R_Key at any given moment. A virtually contiguous range of memory locations can have multiple Regions or Windows associated with it concurrently, each with an associated R_Key.
- A R_Key can be exported to multiple remote responders.
- R_Keys are used only for RDMA and ATOMIC Operations. A R_Key is contained within the packet header.

A responder that supports RDMA and / or ATOMIC Operations shall verify the R_Key, the associated access rights, and the specified virtual address. The responder must also perform bounds checking (i.e. verify that the length of the data being referenced does not cross the associated memory start and end addresses). Any violation must result in the packet being discarded and for reliable services, the generation of a NAK.

9.3.3.3 DMA LENGTH (DMALEN) - 32 BITS

This field indicates the length, in bytes, of the remote DMA operation.

C9-9: For an HCA performing RDMA operations, the minimum length specified in the DMALen field is 0; the maximum length is 2^{31}.

o9-3: If a TCA implements RDMA functionality, the minimum length specified in the DMALen field is 0; the maximum length is 2^{31}.

9.3.4 ATOMIC EXTENDED TRANSPORT HEADER (ATOMICETH) - 28 BYTES

ATOMIC Extended Transport Header (AtomicETH) contains the additional transport fields for ATOMIC Request operations.

Figure 70 ATOMIC Extended Transport Header (AtomicETH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Virtual Address (63-32)			
4-7	Virtual Address (31-0)			
8-11	R_Key			
12-15	Swap (or Add) Data (63-32)			
16-19	Swap (or Add) Data (31-0)			
20-23	Compare Data (63-32)			
24-27	Compare Data (31-0)			

9.3.4.1 VIRTUAL ADDRESS (VA) - 64 BITS

Start address of buffer.

9.3.4.2 R_KEY - 32 BITS

R_Key used to verify remote access to the specified virtual address. See 9.3.3.2 R_Key - 32 bits on page 212.

9.3.4.3 SWAP (ADD) DATA (SWAPDT) - 64 BITS

The data operand used in ATOMIC Operations. In a CmpSwap operation this field is swapped into the addressed buffer if the CmpDt matched the existing buffer contents. In a FetchAdd operation this field is added to the contents of the addressed buffer.

9.3.4.4 COMPARE DATA (CMPDT) - 64 BITS

The data operand used in Compare portion of the CmpSwap ATOMIC Operation.

9.3.5 ACK EXTENDED TRANSPORT HEADER (AETH) - 4 BYTES

ACK Extended Transport Header (AETH) contains the additional transport fields for ACK packets. The ACK Extended Transport header is included in all ACK and the first and last packet of RDMA READ Response messages.

Figure 71 Acknowledge Extended Transport Header (AETH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Syndrome	MSN		

9.3.5.1 SYNDROME

This field indicates if this is an ACK or NAK. If the packet is an ACK and the QP is associated with Reliable Connection transport service, the syndrome also provides the Limit Sequence Number (LSN) - see 9.7.7.2 End-to-End (Message Level) Flow Control on page 314. If packet is a NAK, it indicates the error code. For RNR NAK, this field indicates the responder's requested timer to be used before retransmitting the request.

9.3.5.2 MESSAGE SEQUENCE NUMBER (MSN)

Monotonically increasing (modulo 2^{24}) sequence number of the last message completed at the responder. This field is used to optimize completion processing at the requester.

9.3.5.3 ATOMIC ACKNOWLEDGE EXTENDED TRANSPORT HEADER (ATOMICACKETH) - 8 BYTES

ATOMIC Acknowledge Extended Transport Header (AtomicETH) contains the additional transport fields for ATOMIC response operations.

Figure 72 ATOMIC Acknowledge Extended Transport Header (AtomicAckETH)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Original Remote Data (63-32)			
4-7	Original Remote Data (31-0)			

9.3.5.4 ORIGINAL REMOTE DATA (ORIGREMDT) - 64 BITS

The data result from an ATOMIC Operation. This is the initial contents read from the remote memory buffer.

9.3.6 IMMEDIATE EXTENDED TRANSPORT HEADER (IMMDT) - 4 BYTES

Immediate Data (ImmDt) contains data that is placed in the receive Completion Queue Element (CQE). The ImmDt is only allowed in SEND or RDMA WRITE packets with Immediate Data.

Figure 73 Immediate Extended Transport Header (ImmDt)

bits / bytes	31-24	23-16	15-8	7-0
0-3	Immediate Data			

9.4 TRANSPORT FUNCTIONS

A QP provides the transport layer's client (e.g. the verbs layer in an HCA) with a specific transport service. Different transport services have various reliability levels for connected and connectionless communication. This section describes the basic functions used with each of the transport services. Additional transport sections go into more depth on the specifics of response packets, ordering, error recovery, etc. This section provides the high level view of the functions and how they work.

Not all the functions are available for each transport service, as described in Table 36 below. The Raw Datagram transport service does not use the IBA defined transport functions. Instead, Raw Datagram packets transfer data that is part of some other, non IBA protocol.

Table 36 Transport Functions Supported for Specific Transport Services

Transport Function	Transport Service				
	Reliable Connection	Unreliable Connection	Reliable Datagram	Unreliable Datagram	Raw Datagram
SEND	supported	supported	supported	supported	not applicable
RESYNC	not supported	not supported	supported	not supported	not supported
RDMA WRITE	supported	supported	supported	not supported	not applicable
RDMA READ	supported	not supported	supported	not supported	not applicable
ATOMIC Operations	optional support	not supported	optional support	not supported	not applicable

9.4.1 SEND OPERATION

The SEND Operation is sometimes referred to as a Push operation or as having channel semantics. Both terms refer to how the SW client of the transport service views the movement of data. With a SEND operation the initiator of the data transfer pushes data to the remote QP. The initiator doesn't know where the data is going on the remote node. The remote node's Channel Adapter places the data into the next available receive buffer for that QP. On an HCA, the receive buffer is pointed to by the WQE at the head of the QP's receive queue.

The SEND Operation is referred to as having channel semantics because it moves data much like a mainframe IO channel -- the data is tagged with a discriminator (for IBA the discriminator is the destination LID and QP number) and the destination chooses where to place the data based on the discriminator.

A SEND Operation moves a single message. For the RC, RD, and UC transport services this message may be longer than a single packet. A message may range in size from zero bytes to 2^{31} bytes.

C9-10: The size of a SEND Operation, as generated by a requester, shall be between zero and 2^{31} bytes (inclusive).

C9-11: For RC and UC transport services in an HCA, a request message greater than PMTU in length shall be segmented into PMTU-sized segments for transmission via multiple packets. Similarly, an HCA responder shall reassemble such packets back into a single message.

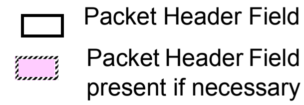
o9-4: For RD transport services in an HCA, a request message greater than PMTU in length shall be segmented into PMTU-sized segments for transmission via multiple packets. Similarly, an HCA responder shall reassemble such packets back into a single message.

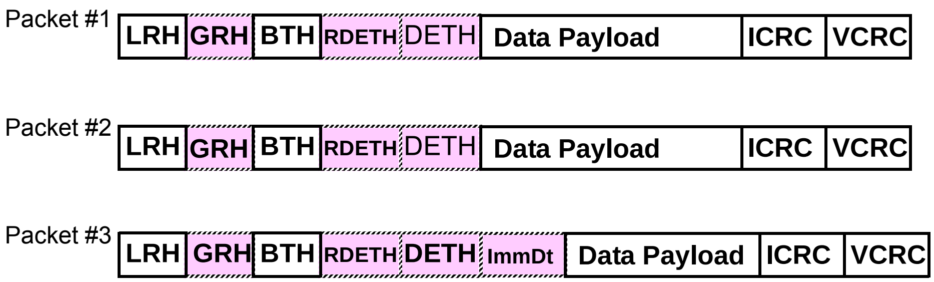
o9-5: For RC, UC and RD transport services in a TCA, a request message greater than PMTU in length shall be segmented into PMTU-sized segments for transmission via multiple packets. Similarly, a TCA responder shall reassemble such packets back into a single message.

C9-12: For the Unreliable Datagram transport service, a SEND Operation shall consist only of single packet messages (i.e. the message data payload is limited to a maximum of the PMTU between the requester and the responder, i.e. 256, 512, 1024, 2048, or 4096 bytes).

A SEND Operation can, at the discretion of the client, include 4 bytes of Immediate data with each send message. If included, the Immediate data is contained within an additional header field (Immediate Extended Transport Header or ImmDt) on the last packet of the SEND Operation.

For example, Figure 74 below shows a SEND Operation of 700 bytes requiring 3 SEND packets, (assuming a 256 Byte PMTU).





	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
DETH	Datagram Extended Transport Header[b]
ImmDt	Immediate Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present only for the Reliable Datagram transport service

b. Present only for Reliable Datagram and Unreliable Datagram transport service

Packet	BTH OpCode[a]
#1	“SEND First”
#2	“SEND Middle”
#3	“SEND Last” or “SEND Last with Immediate”

a. The BTH OpCode determines if the RDETH, DETH, and **ImmDt** headers are present.

A 700 byte SEND Operation uses 3 packets, assuming a 256 Byte PMTU. Acknowledgment Packets, used for reliable transport services, are not shown.

Figure 74 SEND Operation Example

There are several things to note from the above figure:

- The BTH OpCode field determines the start and end of the SEND message.
  - If the SEND message is less than or equal to the PMTU, then the BTH OpCode “SEND Only” or “SEND Only with Immediate” is used.
  - If the SEND message is for a length of zero, then the BTH OpCode “SEND Only” or “SEND Only with Immediate” is used. In this case, there is no Data Payload field, but all other fields are as shown.
  - If the SEND message is greater than the PMTU, then the BTH OpCode of the first packet is “SEND First” and the BTH OpCode of the last packet is “SEND Last” or “SEND Last with Immediate”.
  - If the SEND message is greater than twice the PMTU, then the packets between the first and last use the BTH OpCode “SEND Middle”.

  - Every packet in a message that doesn't have the opcode SEND Only, SEND Only with Immediate, SEND Last, or SEND Last with Immediate shall have a data field of PMTU length.
- The responder node (the destination of the SEND Operation) does not know the final length of the SEND message until the last packet with the “SEND Last” or “SEND Last with Immediate” Op-Code arrives.
- The Packet Sequence Number field is used by the responder to detect out-of-order or missing packets.
- If the entire message is not a multiple of the PMTU, then the initial packets of the message carry a full PMTU number of bytes and the final packet carries the remainder as a partial payload.
- For a given requesting node's QP, once a multi-packet SEND Operation is started, no other request packets may be generated until the “SEND Last” or “SEND Last with Immediate” packet.

C9-13: A multi-packet message shall not be interleaved with other operations on the same SEND Queue.

- Not all SEND messages carry Immediate data. If they do, a special header is included in the last or only packet of the message. The presence of the header is indicated by a special “SEND Last with Immediate Data” or “SEND Only with Immediate Data” Op-Code in the BTH.
- For an HCA, there is no alignment requirement for the source or destination buffers of a SEND message. For buffers within a TCA, any alignment requirement is implementation specific.

The verbs chapter explains how the upper level SW client of an HCA uses a work request to post a buffer that is in turn segmented and sent as packets across the fabric. The same chapter also describes how the destination node posts a receive buffer into which the destination HCA reassembles the data. SEND messages initiated by a TCA use an implementation specific mechanism to create (and respond to) SEND packets.

C9-14: When generating a packet for a SEND operation, the requester shall include at least these headers and fields in every packet of the request: LRH, BTH, Data Payload, ICRC, VCRC.

C9-15: When generating a response to a SEND operation, the responder shall include at least these headers and fields the response: LRH, BTH, AETH, ICRC, VCRC.

9.4.2 RESYNC Operation

A RESYNC operation is supported only for the Reliable Datagram transport service. RESYNC is essentially the same as a zero-length Reliable Datagram SEND-only request but with a several unique properties:

1) RESYNC is used by the requester to force the responder to reset its expected PSN to a value defined by the requester,
2) A RESYNC request carries a data payload of zero length,
3) The responder is required to accept a RESYNC request, even if the currently executing request has not yet completed.
4) A RESYNC request does not, itself, directly consume either a send WQE on the requester side, nor a receive WQE on the responder side.

C9-15.a1: The RESYNC request shall carry a zero length Data Payload.

9.4.3 RDMA WRITE Operation

The RDMA WRITE Operation is used by the requesting node to write into the virtual address space of a destination node. The message may be between zero and 2^{31} bytes (inclusive) and is written to a contiguous range of the destination QP's virtual address space (not necessarily a contiguous range of physical memory).

C9-16: For an HCA requester performing RDMA WRITE operations, the length of an RDMA WRITE message, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

o9-6: If a TCA requester implements RDMA WRITE functionality, the length of an RDMA WRITE message, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

Before allowing incoming RDMA WRITEs, the destination node first allocates a memory range for access by the destination's QP (or group of QPs). A destination's channel adapter associates a 32-bit R_Key with this memory region or window. For a HCA, the verbs layer refers to this as registering a memory region - see 10.6 Memory Management on page 427. TCAs use an implementation-specific mechanism to allocate and manage R_Keys that is outside the scope of the IBA specification.

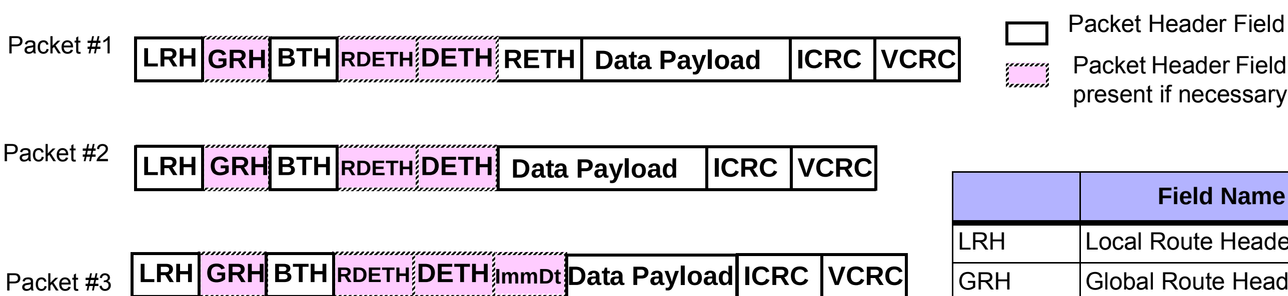
The destination communicates the virtual address, length, and R_Key to any other host it wishes to have access the memory region. The communication of address and R_Key is done by the client upper level protocol - the exchange is outside the scope of the IBA. For example, an application program might embed the address, length, and R_Key into a private data structure that it in turn pushes to other application programs using the SEND Operation.

C9-17: As with SEND Operations, an HCA requester shall segment a RDMA WRITE message larger than the PMTU into multiple packets.

o9-7: If a TCA requester implements RDMA WRITE functionality, it shall segment a RDMA WRITE message larger than the PMTU into multiple packets.

If specified by the verbs layer, Immediate data is included in the last packet of an RDMA WRITE message. The Immediate data is not written to the target virtual address range, but is passed to the client after the last RDMA WRITE packet is successfully processed. E.G. on an HCA the immediate data is placed on the completion queue.

For example, Figure 75 below shows a 700 byte RDMA WRITE (on a path with a 256B PMTU).



	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
DETH	Datagram Extended Transport Header[a]
RETH	RDMA Extended Transport Header
ImmDt	Immediate Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present only for the Reliable Datagram transport service

Packet	BTH OpCode[a]
#1	“RDMA WRITE First”
#2	“RDMA WRITE Middle”
#3	“RDMA WRITE Last” or “RDMA WRITE Last with Immediate”

a. The BTH OpCode field determines if the RDETH, DETH, and ImmDt headers are present.

A 700 byte RDMA WRITE Operation uses 3 packets, assuming a 256 Byte PMTU. Acknowledgment Packets, used for reliable transport services, are not shown.

Figure 75 RDMA WRITE Operation Example

There are several things to note from the above figure:

- The BTH OpCode field determines the start and end of the RDMA WRITE message.
  - If the RDMA WRITE request was for a length of zero, then the BTH OpCode “RDMA WRITE Only” or “RDMA WRITE Only with Immediate” is used. In this case, there is no Data Payload field, but all other fields are as shown.
  - If the RDMA WRITE message is less than or equal to the PMTU, then the BTH OpCode “RDMA WRITE Only” or “RDMA WRITE Only with Immediate” is used.
  - If the RDMA WRITE message is greater than the PMTU, then the BTH OpCode of the first packet is “RDMA WRITE First” and the BTH OpCode of the last packet is “RDMA WRITE Last” or “RDMA WRITE Last with Immediate”.
  - If the RDMA WRITE message is greater than twice the PMTU, then the packets between the first and last use the BTH OpCode “RDMA WRITE Middle”.
  - Every packet in a RDMA WRITE message that doesn’t have the opcode RDMA WRITE Only, RDMA WRITE Only with Immediate, RDMA WRITE Last, or RDMA WRITE Last with Immediate has a data field of PMTU length.
- The RETH header is present in the first (or only) packet of the message. It contains the virtual address of the destination buffer as well as the R_Key and message length fields.
- The Packet Sequence Number field is used by the responder to detect out-of-order or missing packets.
- If the entire message is not a multiple of the PMTU, then the initial packets of the message carry a full PMTU number of bytes and the final packet carries the remainder in a partial payload.
- For a given requesting node’s QP, once a multi-packet RDMA WRITE operation is started, no other request packets may be generated until the “RDMA Last” or “RDMA Last with Immediate Data” packet is sent.

C9-18: For an HCA RDMA WRITE request, a multi-packet message shall not be interleaved with other operations on the same SEND Queue.

o9-8: If a TCA requester implements RDMA WRITE functionality, then for an RDMA WRITE request, a multi-packet message shall not be interleaved with other operations on the same SEND Queue.

- Not all RDMA WRITE messages carry Immediate data. If a RDMA WRITE does, a special header is included in the last (or only) packet of the message. The presence of the header is indicated by a special “RDMA WRITE Last with Immediate Data” or “RDMA WRITE Only with Immediate Data” OpCode in the BTH.

- For an HCA, there is no alignment requirement for the source or destination buffers of an RDMA WRITE message. For buffers within a TCA, any alignment requirement is implementation specific.

C9-19: When generating an RDMA WRITE Request, an HCA requester shall include at least the following headers and fields in each request packet: LRH, BTH, Data Payload, ICRC, VCRC. The first (or only) packet of the request shall also include the RETH.

o9-9: If a TCA requester implements RDMA WRITE functionality, it shall behave as follows. When generating an RDMA WRITE Request, a TCA requester shall include at least the following headers and fields in each request packet: LRH, BTH, Data Payload, ICRC, VCRC. The first (or only) packet of the request shall also include the RETH.

C9-20: When generating an RDMA WRITE Response, an HCA responder shall include at least the following headers and fields in each response packet: LRH, BTH, AETH, ICRC, VCRC.

o9-10: If a TCA responder implements RDMA WRITE functionality, then when generating an RDMA WRITE Response, a TCA responder shall include at least the following headers and fields in each response packet: LRH, BTH, AETH, ICRC, VCRC.

9.4.4 RDMA READ Operation

RDMA READ Operations are similar to RDMA WRITE Operations. They allow the requesting node to read a virtually contiguous block of memory on a remote node. As with RDMA WRITEs, the responding node first allows the requesting node permission to access its memory. The responder passes to the requestor a virtual address, length, and R_Key to use in the RDMA READ request packet.

A single RDMA READ request can read from zero to 2^{31} bytes (inclusive) of data.

C9-21: For an HCA responding to an RDMA READ request, if the requested data size is greater than the PMTU, the responder shall segment the data into PMTU size data segments for transmission as multiple RDMA READ Response packets. The data is reassembled in the requesting node's memory.

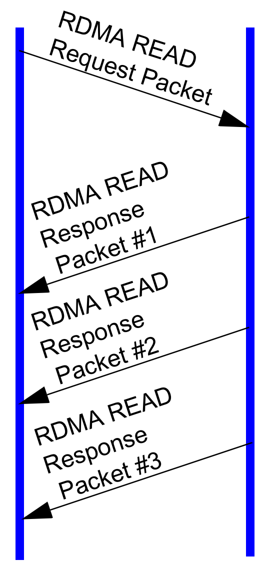
o9-11: If a TCA responder implements RDMA READ functionality, and the requested data size is greater than the PMTU, the responder shall segment the data into PMTU size data segments for transmission as multiple RDMA READ Response packets. The data is reassembled in the requesting node's memory.

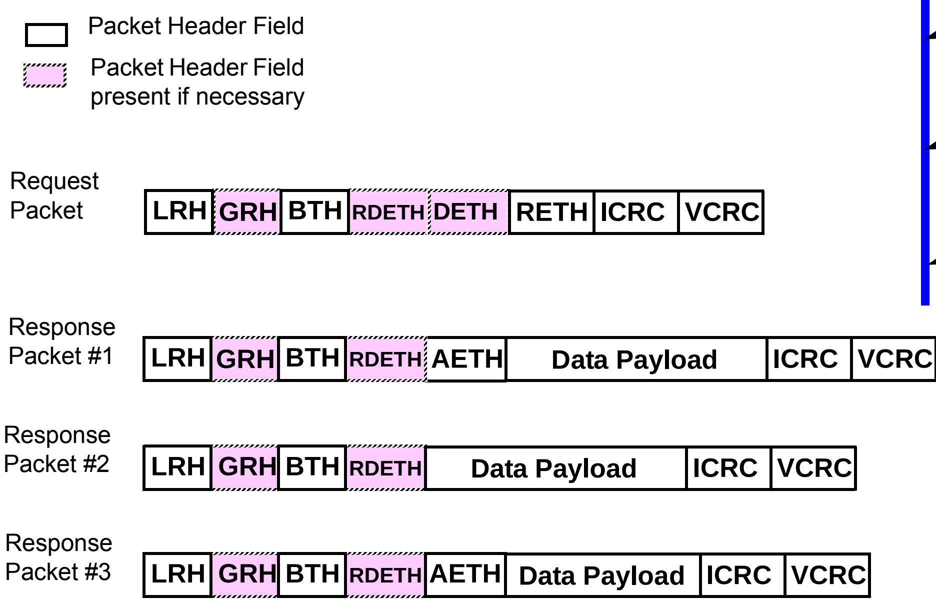
C9-22: For an HCA requester using RDMA operations, the length of the requested RDMA READ data, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

o9-12: If a TCA requester implements RDMA READ functionality, then the length of the requested RDMA READ data, as reflected in the RETH:DMALen field, shall be between zero and 2^{31} bytes (inclusive).

The following example in Figure 76 shows a 700 byte RDMA READ operation (on a path with a 256B PMTU)

A ladder diagram showing the single RDMA READ Request Packet initiated by the requestor node. In this example, the destination node segments the data into three response packets.





	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
DETH	Datagram Extended Transport Header[a]
AETH	Acknowledgment Extended Transport Header
RETH	RDMA Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present only for the Reliable Datagram transport service

Packet	BTH OpCode[a]
Request	"RDMA READ Request"
#1	"RDMA READ Response First"
#2	"RDMA READ Response Middle"
#3	"RDMA READ Response Last"

a. The BTH OpCode field determines if the RDETH and AETH are present.

A 700 byte RDMA READ Operation has 3 response packets, assuming a 256 Byte PMTU.

Figure 76 RDMA READ Operation Example

There are several items to note in the previous figure:

- A single request packet will result in multiple read response packets if the read length is greater than the PMTU.
- The BTH OpCode field identifies the packet as a RDMA READ Request or Response as well as determines if any of the extended transport headers are present.
- The BTH OpCode field determines the start and end of the RDMA READ Acknowledgment message.
  - If the RDMA READ request message requested a zero byte transfer, then the BTH OpCode “RDMA READ Response Only” is used. All other fields remain as shown.
  - If the RDMA READ Acknowledgment message is less than or equal to the PMTU, then the BTH OpCode “RDMA READ Response Only” is used.
  - If the RDMA READ message is greater than the PMTU, then the BTH OpCode of the first packet is “RDMA READ Response First” and of the last packet “RDMA READ Response Last”.
  - If the RDMA READ response message is greater than twice the PMTU, then the packets between the first and last use the BTH OpCode “RDMA READ Response Middle”.
  - Every packet in a RDMA READ Response First or RDMA READ Response Middle message has a data field of PMTU length.
- If the entire message is greater than a multiple of the PMTU, then the initial packets of the response message carry a full PMTU number of bytes and the final packet carries a partial payload.
- The Packet Sequence Number (PSN) field is used to detect out-of-order or missing response packets.
- After initiating a RDMA READ Request packet, the requesting node may send out additional request packets without waiting for the response packets to return. See section 9.7.3.1 Requester Side - Generating PSN on page 256 for an explanation of how the PSN is determined for subsequent request packets.
- The maximum number of RDMA READ Requests for a particular QP that can be outstanding at any one time is negotiated at connection establishment time. A responder may restrict the connection to as few as one outstanding RDMA READ request per QP. If ATOMIC Operations are supported, the number of outstanding requests negotiated at connection establishment time includes both ATOMIC Operation requests and RDMA READ requests.
- RDMA READ packets never carry Immediate data.

RDMA READ Requests are retried if the requester did not receive the proper response.

- Retried RDMA READ Requests need not start at the same address nor have the same length as the original RDMA READ. The retried request may only reread those portions that were not successfully responded to the first time.
- The responder validates the R_Key and RDMA READ virtual address for the retried request.
- The PSN of the retried RDMA READ must be in the duplicate PSN region. See Section 9.7.1 Packet Sequence Numbers (PSN) on page 248
- The PSN of the retried RDMA READ request need not be the same as the PSN of the original RDMA READ request. Any retried request must correspond exactly to a subset of the original RDMA READ request in such a manner that all potential duplicate response packets must have identical payload data and PSNs regardless of whether it is a response to the original request or a retried request.
- For an HCA, there is no alignment requirement for the source or destination buffers of an RDMA READ message. For buffers within a TCA, any alignment requirement is implementation specific.

C9-23: When generating an RDMA READ Request, an HCA requester shall include at least the following headers and fields in its request packet: LRH, BTH, RETH, ICRC, VCRC.

o9-13: If a TCA requester implements RDMA operations, then it shall include at least the following headers and fields in its request packet: LRH, BTH, RETH, ICRC, VCRC.

C9-24: When generating an RDMA READ Response, an HCA responder shall include at least the following headers and fields in each response packet: LRH, BTH, Data Payload, ICRC, VCRC. If the response packet BTH:Opcode is “RDMA READ Response First, RDMA READ Response Last, or RDMA READ Response Only, the packet shall also include an AETH. If the response packet BTH:Opcode is “RDMA READ Response Middle, an AETH shall not be included.

o9-14: If a TCA responder implements RDMA operations, then it shall include at least the following headers and fields in each response packet: LRH, BTH, Data Payload, ICRC, VCRC. If the response packet BTH:Opcode is “RDMA READ Response First, RDMA READ Response Last, or RDMA READ Response Only, the packet shall also include an AETH. If the response packet BTH:Opcode is “RDMA READ Response Middle, an AETH shall not be included.

9.4.5 ATOMIC OPERATIONS

ATOMIC Operations execute a 64-bit operation at a specified address on a remote node. The operations atomically read, modify and write the destination address and guarantee that operations on this address by other QPs on the same CA do not occur between the read and the write. The scope of the atomicity guarantee may optionally extend to other CPUs and HCAs.

ATOMIC Operations use the same remote memory addressing mechanism as RDMA READs and Writes. The virtual address specified in the request packet is in the address space of the remote QP that the ATOMIC Operation has targeted.

ATOMIC Operations consist of two packet types, the “ATOMIC Command”, request packet and the “ATOMIC Acknowledge” response packet.

1) ATOMIC Operations are only supported by the Reliable Connection and Reliable Datagram transport services.
2) ATOMIC Operations do not support Immediate data.
3) ATOMIC Operations support is strongly recommended to be provided strictly in hardware.[1]
4) The virtual address in the ATOMIC Command Request packet shall be naturally aligned to an 8 byte boundary. The responding CA checks this and returns an Invalid Request NAK if it is not naturally aligned.

IBA defines the following ATOMIC Operations:

- **FetchAdd** (Fetch and Add)

  The FetchAdd ATOMIC Operation tells the responder to read a 64-bit buffer value at a naturally aligned virtual address in the responder's memory, perform an unsigned[2] add using the 64-bit Add Data field in the AtomicETH, and write the result (must match the memory type at the requester) back to the same virtual address. The responder's operation shall be atomic (i.e. undisturbed by other entities) per section 9.4.5.1 Atomicity Guarantees on page 229.

  The FetchAdd operation is performed in the endian format of the target memory. The original remote data is converted from the endian format of the target memory for return. The fields are in Big-endian format on the wire.

1. CA implementations may use software assists - this shall be indistinguishable from a hardware-only implementation; Performance must be such that higher level software applications are not affected.

2. If Signed numbers are used, this is the same as using twos complement arithmetic (the carry is not saved nor reported).

The requestor specifies:

- Remote data address and R_Key
- Add data

The acknowledge packet returns:

- Original remote data

After the operation, the responder's memory at the specified virtual address contains the unsigned sum of the original value and the Add field in the AtomicETH header. All operations on the requester's memory are done in the native endian format of the requester.

- **CmpSwap** (Compare and Swap)

The CmpSwap ATOMIC Operation tells the responder to read a 64-bit value at a naturally aligned virtual address in the responder's memory, compare it with the Compare Data field in the AtomicETH header, and, if they are equal, write the Swap Data field from the AtomicETH header into the same virtual address. If they are not equal, the contents of the responder's memory are not changed. In either case, the original value read from the virtual address is returned to the requester. The responder's operation shall be atomic (i.e. undisturbed by other entities) per section 9.4.5.1 Atomicity Guarantees on page 229.

The requestor specifies:

- Remote data address and R_Key
- Write (swap) data
- Compare data

The acknowledgment packet returns:

- Original remote data

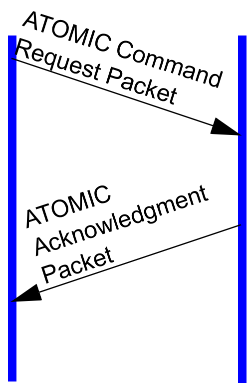
After the operation, the remote data buffer contains the "original remote value" (if comparison did not match) or the "Write (swap) data" (if the comparison did match).

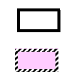
The CmpSwap operation involves three 8 byte data buffers, the compare data, the write (swap) data, and the original remote data. All three are transmitted within the request and response packets in byte big endian format. All operations on the responder's CA memory are done in the native endian format of that memory system. All operations on the requestor's memory are done in the native endian format of the requestor.

For example, consider a big endian CA initiating a CmpSwap ATOMIC Operation request packet to a little endian responder. The request packet contains two big endian data fields: the compare data and the write (swap) data. The responder converts these data fields to little endian format and does the compare and swap operation. The original target data field is converted to big endian format and returned in the response packet.

.

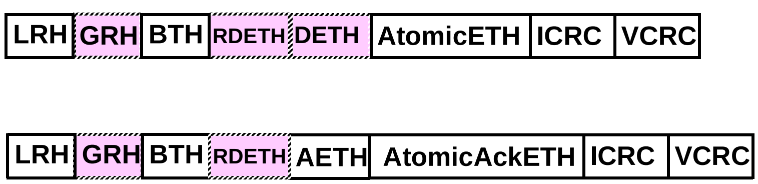
A ladder diagram showing the “ATOMIC Command” Request Packet and the returning “ATOMIC Acknowledge” response





Request Packet

Acknowledgment Packet



	Field Name
LRH	Local Route Header
GRH	Global Route Header
BTH	Base Transport Header
RDETH	Reliable Datagram Extended Transport Header[a]
AETH	Acknowledgment Extended Transport Header
DETH	Datagram Extended Transport Header[a]
AtomicETH	ATOMIC Request Extended Transport Header
AtomicAck-ETH	ATOMIC Acknowledgment Extended Transport Header
ICRC	Invariant CRC
VCRC	Variant CRC

a. Present in ATOMIC Operations only for the Reliable Datagram transport service

Figure 77 ATOMIC Operation Example

o9-15: When generating an ATOMIC Operation request, a requester shall include at least an LRH, a BTH, an AtomicETH, an ICRC and a VCRC. The sources of data for the LRH, BTH and AtomicETH headers shall be as shown in Table 60 Packet Fields and Parameters by Service on page 386.

o9-16: When responding to an ATOMIC Operation request, a responder shall include in its response packet at least an LRH, BTH, AETH, AtomicAckETH, ICRC and a VCRC.

9.4.5.1 ATOMICITY GUARANTEES

o9-17: Atomicity of the read/modify/write on the responder’s node by the ATOMIC Operation shall be assured in the presence of concurrent atomic accesses by other QPs on the same CA.

o9-18: A CA may optionally assure atomicity of ATOMIC Operations in the presence of concurrent memory accesses from other CAs, IO devices, and CPUs. For a HCA, the Verbs layer shall report whether it supports this enhanced atomicity guarantee.

9.4.5.2 ATOMIC ACKNOWLEDGMENT GENERATION AND ORDERING RULES

1) For the requestor, an ATOMIC Operation is considered complete when the response packet returns.
2) If an RDMA READ work request is posted before an ATOMIC Operation work request then the atomic may execute its remote memory operations before the previous RDMA READ has read its data. This can occur because the responder is allowed to delay execution of the RDMA READ. Strict ordering can be assured by posting the ATOMIC Operation work request with the fence modifier. See the description for the fence modifier Post Send Request. The fence modifier causes the requestor to wait till the RDMA READ completes before issuing the ATOMIC Operation.
3) When a sequence of requests arrives at a QP, the ATOMIC Operation only accesses memory after prior (non-RDMA READ) requests access memory and before subsequent requests access memory. Since the responder takes time to issue the response to the atomic request, and this response takes more time to reach the requestor and even more time for the requestor to create a completion queue entry, requests after the atomic may access the responders memory before the requestor writes the completion queue entry for the ATOMIC Operation request.
4) Each ATOMIC Operation request requires an explicit response and acknowledge message. An ATOMIC Operation response, with a properly formed AETH, is considered an acknowledge message.

9.4.5.3 ERROR BEHAVIOR

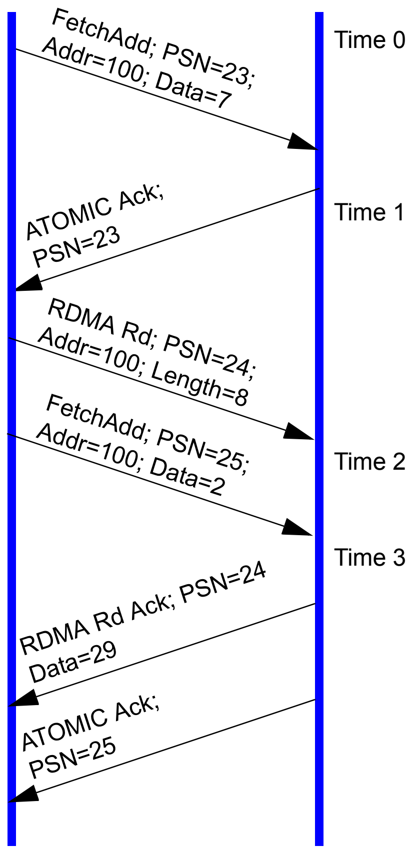
A responder utilizes vendor specific resources and facilities to implement ATOMIC Operations and RDMA READs as well as to facilitate retried ATOMIC requests. It is the responsibility of the requestor to ensure that all unacknowledged ATOMIC operations and RDMA READs combined do not overrun the receiver resources. The number of these resources is negotiated on a per QP basis at connection setup (see 9.4.4 RDMA READ Operation on page 222 and 9.4.5 ATOMIC Operations on page 227).

The responding node saves the reply data, the PSN, and an indication that the stored data is from an ATOMIC Operation. This saved data is used to generate the response for retried ATOMIC Operations. Note that the execution of an RDMA READ operation may consume the same resources as is used to save the ATOMIC Operation PSN and reply data. The information is stored in the destination QP's “connection context”. The “connection context” is the QP context for Reliable Connection Service. For Reliable Datagram Service, the “Connection context” is actually the “EE context”.

Several rules determine when the responder stores the PSN and reply data of an ATOMIC Operation:

- Only valid, new ATOMIC Operation requests (i.e. all header checks are valid, the incoming PSN matches the expected PSN, the R_Key is valid for the data being accessed, and the address is aligned to a 64b boundary) are saved.
- If the responder QP supports multiple outstanding ATOMIC Operations and RDMA READ Operations, the information on each valid request is saved in FIFO order. The FIFO depth is the same as the maximum number of outstanding ATOMIC Operations and RDMA READ requests negotiated on a per QP basis at connection setup.
- Repeated ATOMIC or RDMA READ Operations are not saved again.

The saved ATOMIC and RDMA READ state is shown in the figure below.



Timeline of Responder's State

Time	Contents of Memory at address 100	Per QP State[a] Tracking Most Recent ATOMIC & RDMA READ Operations					
		Most Recent			2nd Most Recent		
		Op	PSN	Result	Op	PSN	Result
Time 0	20	na	na	na	na	na	na
Time 1	27	ATOMIC	23	20	na	na	na
Time 2	27	RDMA READ	24	na	ATOMIC	23	20
Time 3	29	ATOMIC	25	27	RDMA READ	24	na

a. For Reliable Connection Service, the state bits for tracking the most recent ATOMIC and RDMA READ Operations are kept in the per QP State. For Reliable Datagram Service these state bits are kept in the EE Context instead of the Per QP State.

The ladder diagram shows multiple ATOMIC and RDMA READ Requests. In this example the responder's QP has agreed at connection setup time that it can accept up to any combination of 2 outstanding ATOMIC or RDMA READ Operations. This example shows how the responder maintains state of the recent ATOMIC and RDMA READ operations. Note also in the example a RDMA READ preceding an ATOMIC Operation but targeting the same address may return the value after the ATOMIC executes. Strict ordering is possible in an HCA by using the "fence" option when posting the ATOMIC following the RDMA READ. See 11.4.1.1 Post Send Request on page 525

Figure 78 Responder State Maintained for ATOMIC & RDMA READ Operations

An ATOMIC Operation is guaranteed to execute at most once. If the ATOMIC Operation does not execute on the destination, it is reported to the sender (e.g. an R_Key protection fault) with the appropriate NAK syndrome response.

However, like all operations, a non-recoverable error that occurs after execution at the responder, but before the response reaches the requester (e.g. a fatal HCA error), results in the requester not knowing the state of the responder's memory. This case must be detected and dealt with by the client or upper layer protocol.

As with all operations, errors could occur on any of the transfers. If the original "ATOMIC Command" request is lost, or the "ATOMIC Acknowledge" is lost, the sender will retry using the normal retry procedures. If the retry fails, it is not certain whether the ATOMIC Operation took place at the destination, but the connection will be in the Error state.

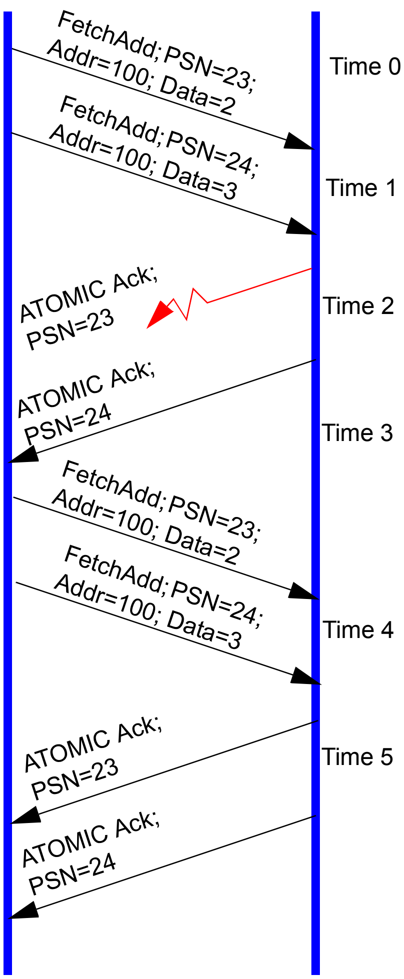
As with retries of Send and RDMA WRITE operations, if the responding CA has actually executed the request, it will only acknowledge the request again, not re-run the ATOMIC Operation. This is necessary since an ATOMIC Operation is not idempotent. The responder recognizes a retried ATOMIC Operation and returns the reply data from the original acknowledgment that was previously stored in the QP (or EE context for Reliable Datagram service) “hidden state”. The responder returns the stored result of an ATOMIC Operation if the following conditions are met:

- The request is valid (i.e. header an OpCode are valid)
- The request is for an ATOMIC Operation (the responder may check the ATOMIC Operation OpCode is the same as that of the stored operation)
- The PSN of the request is in the “duplicate region”. See a description of the PSN space in Section 9.7.1 Packet Sequence Numbers (PSN) on page 248.
- The PSN matches that of a saved ATOMIC Operation.

A retried ATOMIC Operation that does not meet the above conditions is discarded by the responder. See Table 58 Responder Error Behavior Summary on page 375

When an ATOMIC Operation is retried, the responder does not validate the R_Key nor does it translate the virtual address in the retried request.

The figure below demonstrates a failed ATOMIC Operation response packet and shows the retried request and the eventual successful response.



Timeline of Responder's State

Time	**Contents of Memory at address 100**	**Per QP State[a] Tracking Most Recent ATOMIC & RDMA READ Operations**					
		Most Recent			**2nd Most Recent**		
		Op	**PSN**	**Result**	**Op**	**PSN**	**Result**
Time 0	20	na	na	na	na	na	na
Time 1	22	ATOMIC	23	20	na	na	na
Time 2	25	ATOMIC	24	22	ATOMIC	23	20
Time 3	25	ATOMIC	24	22	ATOMIC	23	20
Time 4	25	ATOMIC	24	22	ATOMIC	23	20
Time 5	25	ATOMIC	24	22	ATOMIC	23	20

a. For Reliable Connection Service, the state bits for tracking the most recent ATOMIC and RDMA READ Operations are kept in the per QP State. For Reliable Datagram Service these state bits are kept in the EE Context instead of the Per QP State.

The ladder diagram shows multiple ATOMIC and RDMA READ Requests. In this example the responder's QP has agreed at connection setup time that it can accept up to any combination of 2 outstanding ATOMIC or RDMA READ Operations. This example shows a lost ATOMIC acknowledgment (at Time 2). When the request is retried, the original result value is returned. The original value is returned even if subsequent operations from the same or a different QP have modified the target of the ATOMIC Operation.

Figure 79 Retrying ATOMIC Operations

If all retries fail, that implies that the connection is lost, and the error recovery routines in the requesting CA's driver will inform the local application.

The size of the operation is always 64-bits. The target must be naturally aligned (low 3 bits of the virtual address must be zero). An error will be reported if the R_Key range does not fully enclose the target. If this or another protection error occurs, it will be reported (NAK_Remote_Access) but will not result in taking any of the "ATOMIC Operation hidden queue" resources. That is, if the same request is repeated (same PSN) and the responding side has subsequently allocated an R_Key range, this new operation will now succeed.

9.4.6 Reserved and Manufacturer Defined Transport Function OpCodes

The IBA has two mechanisms for future expansion of its transport layer:

- Reserved and Manufacturer Defined BTH OpCodes

IBA Transport layer functionality can be expanded by defining new BTH OpCodes. Two blocks of undefined OpCodes are specified. One for future revisions of the IBA and one block for manufacturer specific functions.

9.5 Transaction Ordering

This section defines the rules for ordering of transmission, execution, and completion for transactions for a given QP:

C9-25: A requester shall transmit request messages in the order that the Work Queue Elements (WQEs) were posted.

C9-26: For messages that are segmented into PMTU-sized packets, the data payload shall use the same order as the data segments defined by the WQE.

Packets from a given source QP to a given destination QP travel on the same path through the fabric and are received in the same order they were injected.

C9-27: For reliable services on an HCA, all acknowledge packets shall be strongly ordered, e.g. all previous RDMA READ responses and ATOMIC responses shall be injected into the fabric before subsequent SEND, RDMA WRITE responses, RDMA READ response or ATOMIC Operation responses.

o9-19: If a TCA responder implements Reliable Connection service, or if a CA responder implements Reliable Datagram service, all acknowledge packets shall be strongly ordered. That is, all previous RDMA READ responses and ATOMIC responses shall be injected into the fabric before subsequent SEND, RDMA WRITE responses, RDMA READ response or ATOMIC Operation responses.

C9-28: A responder shall execute SEND requests, RDMA WRITE requests and ATOMIC Operation requests in the message order in which they are received. If the request is for an unsupported function or service, the appropriate response (for example, a NAK message, silent discard, or logging of the error) shall also be generated in the PSN order in which it was received.

- An application shall not depend upon the order of data writes to memory within a message. For example, if an application sets up data buffers that overlap, for separate data segments within a message, it is not guaranteed that the last sent data will always overwrite the earlier.

C9-29: The completion at the receiver is in the order sent (applies only to SENDs and RDMA WRITE with Immediate) and does not imply previous RDMA READs are complete unless fenced by the requester.

C9-30: A requester shall complete WQEs in the order in which they were transmitted.

C9-31: A work request with the fence attribute set shall block until all prior RDMA Read and Atomic WRs have completed.

C9-32: All WQEs shall be completed in the order they were posted independent of their execution order.

Due to the ordering rule guarantees of requests and responses for reliable services, the requester is allowed to write CQ completion events upon response receipt.

o9-20: An application shall not depend of the contents of an RDMA WRITE buffer at the responder until one of the following has occurred:

- Arrival and Completion of the last RDMA WRITE request packet when used with Immediate data.
- Arrival and completion of a subsequent SEND message.
- Update of a memory element by a subsequent ATOMIC operation.

o9-21: An application shall not depend on the contents of an RDMA READ target buffer at the requestor until the completion of the corresponding WQE.

o9-21.a1: An application shall not depend upon the contents of a SEND buffer at the responder until it has been completed.

C9-33: An application shall not depend on the contents of a receive queue buffer until the corresponding receive WQE has been completed.

9.6 Packet Transport Header Validation

Packet transport header validation is conducted on each packet that is passed up to the transport layer from the lower IBA layers. The purpose is to ascertain that the inbound packet can be associated with a particular queue pair. If it cannot, the packet is silently discarded. Packet transport header validation applies only to packets using the IBA transport.

C9-34: The transport layer shall validate the packet headers of all packets using the IBA transport according to the requirements in this section (9.6 Packet Transport Header Validation on page 236. A packet shall be deemed to be using the IBA transport if the msb of the LRH:LNH field is set to 1. If the msb of the LRH:LNH field is set to zero, then the packet is a raw packet. Raw packets are described in Section 9.8.4 Raw datagrams on page 360.

C9-35: For each inbound packet using the IBA transport, a CA shall validate the packet according to the state diagram shown in Figure 80[1]. The details of the state diagram are discussed in the remainder of this section.

If the packet can be associated with a given queue pair, further validation is conducted by comparing certain characteristics of the packet with context information stored with the queue pair (or EE Context, in the case of reliable datagram service). This level of validation is described in Section 9.7 Reliable Service on page 247 and Section 9.8 Unreliable Service on page 341.

Throughout this section, the phrase “packet is silently dropped” is used. The responder, unless otherwise noted, behaves as follows when a silent drop occurs:

- No acknowledge message is returned.
- No receive WQE is consumed by the responder.
- The errant request packet is not executed.
- Any request packets received prior to the errant request are executed and completed normally.
- Responder does not update its expected PSN.
- Responder resumes waiting for a valid inbound request packet.

The requester, unless otherwise noted, behaves as follows when a silent drop occurs:

1. The LVer field of the LRH is verified in the link layer before a packet is presented to the transport layer. The ICRC and VCRC headers are also verified in the link layer before a packet is presented to the transport layer. A packet with an invalid LVer field, invalid ICRC or invalid VCRC is dropped silently before reaching the transport layer.

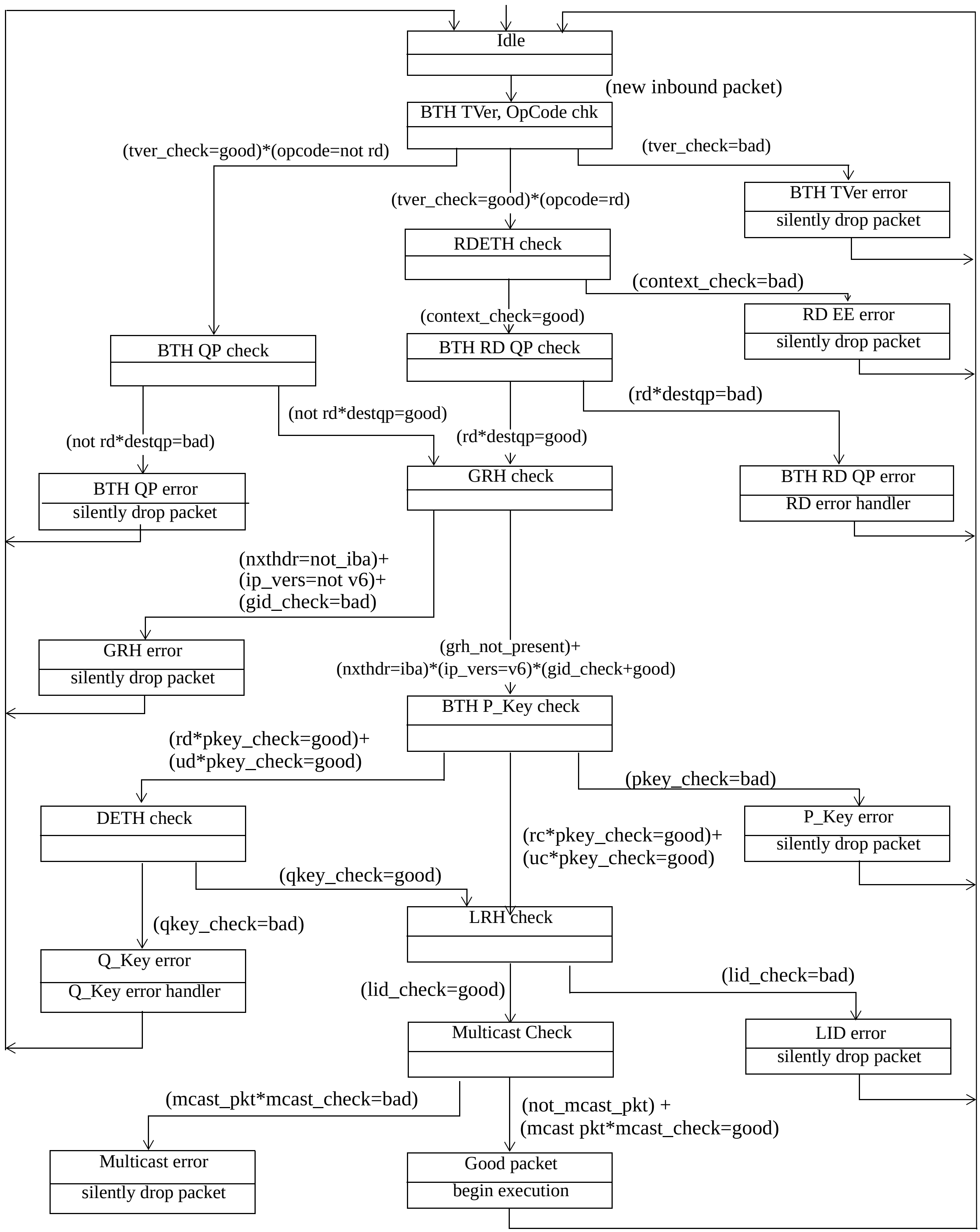


Figure 80 Packet Header Validation Process

- No send WQEs are completed as a result of a packet that is silently dropped.
- No direct action is taken as a result of the silently dropped packet, although error counters may be incremented or other similar events may occur.
- The silently dropped packet shall not count for purposes of satisfying the transport timer.

The queue state is not be changed. In addition, for connected transport services or reliable datagram, the connection or EE context is not torn down.

9.6.1 VALIDATING HEADER FIELDS

This section specifies the headers and fields that must be validated by a receiver of an inbound packet (either a request or response packet) before it can rely on the integrity of the packet.

9.6.1.1 BTH CHECKS

This section describes the fields of the BTH that must be validated for all incoming packets.

9.6.1.1.1 BTH:TVER VALIDATION

C9-36: The transport shall verify that the version number of the transport headers is supported by the CA or router. If the CA, switch or router does not support the indicated version, the packet shall be silently dropped. The only valid transport version is zero.

tver_check

- good: TVer field of BTH is 0x0
- bad: TVer field of BTH is non-zero

9.6.1.1.2 BTH:DESTINATION QP, OPCODE CHECK

Since the OpCode contained in the BTH of the inbound packet is used to determine if the selected destination QP is valid, OpCode validation is combined with validating the destination QP and its current condition.

C9-37: The transport shall verify that the destination QP exists and that the QP state is valid for receiving the inbound packet.

o9-22: If the CA implements RD service, the transport layer shall verify the destination QP is valid for the CA or router and that the QP state is valid for receiving the inbound packet. If the destination QP or its state is invalid, the response shall depend on the EE Context. If the EE Context is valid, a NAK-invalid RD request must be returned; else if the EE context is invalid, the packet shall be silently dropped.

o9-23: For CAs which support Unreliable Datagram Multicast, the destination QP value of 0xFFFFFF shall only be valid if there is at least one locally managed QP which is configured for IBA Unreliable Datagram Multicast service.

C9-38: BTH:OpCode[7:5] shall be checked to ensure that the service requested (RC, UC, RD, UD) is consistent with the configuration of the destination QP.

The response to an inbound packet which contains either an invalid destination QP, or whose destination QP is not in a valid state for receiving the inbound packet, is dependent on the service being requested. This is determined by examining BTH:OpCode[7:5], which indicates whether the requested service is RC, RD, UC or UD.

Furthermore, if BTH:OpCode[7:5] indicates that the packet is for RD service, then the remainder of the OpCode bits must be examined to determine if the inbound packet is a request or a response packet.

C9-39: If BTH:OpCode[7:5] indicates that the packet is for RC, UC or UD services, and the destination QP does not exist, or the destination QP is not configured to provide the requested service, or the destination QP state is invalid, then the inbound packet is silently dropped.

C9-40: If BTH:OpCode[7:0] indicates an RD request packet, (SEND, RDMA READ Request, etc.), and the destination QP does not exist, or the destination QP is not configured to provide RD service, or the destination QP state is invalid, then a NAK-Invalid RD Request shall be returned. If BTH:OpCode[7:0] indicates an RD response packet (RDMA READ Response, Acknowledge, etc.), and the destination QP does not exist, or the destination QP is not configured to provide RD service, or the destination QP state is invalid, then the inbound packet shall be silently dropped.

Table 37 Verification of Destination QP

Error Condition	Description
Invalid Destination QP identifier	No such QP exists on this CA. If the QP identifier is the IBA unreliable multicast QP (0xFFFFFF), there is no QP configured for IBA unreliable multicast service on this CA.
Incorrect Destination QP Configuration	The destination QP configuration is inconsistent with the service requested by BTH:OpCode[7:4].
Request packet: QP is not in Ready-to-Send state, Send-Queue-Drain state, or Ready-to-Receive state.	Receive queue is not in a proper state to accept an inbound request packet.
Acknowledge packet: QP is not in Send-Queue-Drain state or Ready-to-Send state.	Send queue is not in a proper state to accept an inbound response packet.

destqp_check

- good: destination QP specified in BTH is a valid QP, and it is in the correct state to receive the packet, and the configuration of the QP is consistent with the service being requested.
- bad: destination QP specified in BTH does not exist, or is not in the correct state to receive the packet, or is configured inconsistently with the service being requested.

9.6.1.1.3 BTH:P_KEY

C9-41: If the destination QP is QP0, the BTH:P_Key shall not be checked.

C9-42: If the destination QP is QP1, the BTH:P_Key shall be compared to the set of P_Keys associated with the port on which the packet arrived. If the P_Key matches any of the keys associated with the port, it shall be considered valid.

C9-43: For all destination QPs other than QP0 or QP1, for all transport services except Reliable Datagram, the P_Key shall be compared with the P_Key associated with the responder's receive queue. An invalid P_Key shall cause the request packet to be silently dropped.

o9-24: For Reliable Datagram, the P_Key shall be compared with the P_Key associated with the responder's EE Context. An invalid P_Key shall cause the request packet to be silently dropped.

pkey_check

- good: BTH:P_Key matches value associated with recv queue or EE Context
- bad: BTH:P_Key does not match value associated with recv queue or EE Context

9.6.1.2 GRH CHECKS

This sections describes the fields of the GRH, if present, that must be validated.

As specified in Section, 9.6.1.5.2 IBA Unreliable Multicast Checks on page 246, a multicast packet must include a GRH.

9.6.1.2.1 GRH:NEXT HEADER

C9-44: If there is a GRH present, the Next Header field of the GRH must be checked. The value of the Next Header field should be set to 0x1B. Any other value indicates that this packet does not use the IBA transport, and the packet shall be silently dropped.

nxthdr_check

- good: GRH:NxtHdr field indicates IBA transport
- bad: GRH:NxtHdr field indicates non-IBA transport

9.6.1.2.2 GRH:IPVers

C9-45: If there is a GRH present, the version field of the GRH shall be checked. If the version number is anything other than 6, the packet shall be silently dropped.

ip_vers

- not_v6: invalid GRH version number
- v6: GRH version number is valid

9.6.1.2.3 GRH:SGID, GRH:DGID

Connection Management is responsible for loading the primary SGID and DGID in the transport layer. If the given CA supports automatic path migration, a set of alternate SGID and DGID are also loaded. Primary and alternate GID comparison and actions are per the rules defined in Section 17.2.8 Automatic Path Migration on page 836.

If a GRH is present, the SGID and DGID shall be verified as follows:

C9-46: If the destination QP is configured for UD transport service, the SGID shall not be validated at the transport layer. The DGID shall only be validated if the packet is a valid multicast packet. See 9.6.1.5.2 IBA Unreliable Multicast Checks on page 246 for a definition of a valid multicast packet.

C9-47: If the destination QP is configured for RC, UC, or RD transport services, the SGID and the DGID shall be validated at the transport layer. For RC and UC, invalid packets must be silently dropped. For RD, a “NAK Invalid RD Request” must be returned for invalid packets.

The DGID is validated as follows:

1) If the DGID is set to the Reserved GID, the DGID is invalid.
2) If the DGID is set to the Loopback GID, the DGID is invalid.
3) If the DGID’s scope indicates a Multicast GID but there is no locally associated QP, then the DGID is invalid.

Following these checks, the DGID is compared against the following. If it matches none of these, then the DGID is invalid.

4) The DGID is compared against the Primary DGID.
5) The upper 64-bits of the DGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the DGID is compared with the lower 64-bits of the Primary DGID

6) If Automatic path migration is supported, the DGID is compared with the Alternate DGID.

7) If Automatic path migration is supported, the upper 64-bits of the DGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the DGID is compared with the lower 64-bits of the Alternate DGID

The SGID is validated as follows:

1) If the SGID is set to multicast, the SGID is invalid.

2) If the SGID is set to the Loopback GID, the SGID is invalid.

Following these checks, the SGID is compared to the following. If the SGID does not match at least one of the following, it is invalid.

3) The SGID is compared against the Primary SGID

4) The upper 64-bits of the SGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the SGID is compared with the lower 64-bits of the Primary SGID

5) If Automatic path migration is supported, the SGID is compared with the Alternate SGID

6) If Automatic path migration is supported, the upper 64-bits of the SGID is compared against the default GID prefix (0xFE80::0) and the lower 64-bits of the SGID is compared with the lower 64-bits of the Alternate SGID

gid_check

- good GRH SGID and DGID compared successfully
- bad GRH SGID or DGID is invalid

9.6.1.3 RDETH CHECKS

The section describes the fields of the RDETH, if present, that must be validated for reliable datagram service.

9.6.1.3.1 RDETH:EE CONTEXT

o9-25: If BTH:OpCode[7:5] indicates RD transport service, the RDETH shall be validated.

o9-26: The EE Context Identifier shall be verified per the rules in Table 38 Verification of EE Context on page 244. If the EE context is invalid, the packet must be silently dropped.

Table 38 Verification of EE Context

Error Condition	Description
Invalid Destination EE Context identifier	No such EE Context exists on this CA.
Request packet: EE Context is not in Ready-to-Send state, Send-Queue-Drain state or Ready-to-Receive state.	EE Context is not in a proper state to accept an inbound request packet.
Acknowledge packet: EE Context is not in Ready-to-Send state or Send-Queue-Drain state.	EE Context is not in a proper state to accept an inbound response packet.

context_check

- good: EE Context specified in RDETH is valid
- bad: EE Context specified in RDETH is invalid

9.6.1.4 DETH CHECKS

This section describes the fields of the DETH, if present, that must be checked for datagram service, either reliable or unreliable.

9.6.1.4.1 DETH:Q_KEY

C9-48: If the destination QP is QP0, the DETH:Q_Key field shall not be validated.

C9-49: If the destination QP is QP1, the DETH:Q_Key field shall be considered valid if it compares successfully to the well-known Q_Key 0x80010000.

C9-50: For all packets received for a queue pair configured for datagram service, except QP0, the Q_Key shall be checked by the receiver's receive queue. If the Q_Keys do not match, the responder's behavior depends on whether the service is unreliable datagram or reliable datagram and shall be as follows:

Unreliable Datagram: the packet shall be silently dropped.

Reliable Datagram:

- A NAK-Invalid RD Request shall be returned.
- The P_Key used in the NAK may be supplied by the responder's EE Context or it may be extracted from the request packet being acknowledged.

- The PSN used in the NAK message is the PSN of the errant request packet.
- The EE Context's PSN is unchanged; it remains pointing to the failed request packet.
- The responder resumes waiting for a valid inbound request packet.

C9-51: The responder must not return an acknowledge message for a packet until the Q_Key and the P_Key for the packet have been checked by the receive queue.

qkey_check

- good: the Q_Key contained in the BTH matches that associated with the receive queue's stored Q_Key.
- bad: the Q_Key contained in the BTH does not match that associated with the receive queue's stored Q_Key.

9.6.1.5 LRH CHECKS

This section describes the fields of the LRH that must be validated for all inbound packets.

9.6.1.5.1 LRH:SLID, LRH:DLID

C9-52: The 16 bit fully resolved SLID and DLID contained in the LRH shall be validated.

C9-53: The DLID shall be validated only for reliable connected, unreliable connected or reliable datagram service. The DLID shall not be validated for Unreliable Datagram service.

C9-54: The SLID shall be validated only for reliable connected, unreliable connected or reliable datagram service. The SLID shall not be validated for Unreliable Datagram service.

To be valid, the SLID and the DLID contained in the LRH must compare exactly to one of the following:

1) Permissive LID
2) Multicast LID (for DLID only)
3) Primary LID
4) Alternate LID

C9-55: The permissive LID shall only be accepted as valid if the destination queue pair is QP0.

C9-56: If the SLID is a multicast LID, it shall be invalid.

C9-57: In an HCA configured for Reliable Connection or Unreliable Connection service, if an invalid LID is detected, the packet shall be silently dropped. For RC and UC service, this check is performed by the send or receive queue.

o9-27: If a TCA implements Reliable Connection or Unreliable Connection service and an invalid LID is detected, the packet shall be silently dropped. For RC and UC service, this check is performed by the send or receive queue.

o9-28: If a CA implements Reliable Datagram service, and if an invalid LID is detected, the packet shall be silently dropped. For RD service, this check is performed by the EE Context.

The primary SLID and DLID are stored in the QP or EE Context. If the given channel adapter supports transparent migration, an alternate SLID and DLID are also stored in the QP or EE Context as part of the alternate path. The choice of whether to compare the inbound SLID and DLID to the primary or alternate LIDs is a function of the current state of the automatic path migration state machine and the state of the MigReq bit in the BTH.

lid_check

- good: SLID and DLID contained in the LRH matches the SLID and DLID, respectively, stored in the QP or EE Context.
- bad: SLID or DLID contained in the LRH does not match the SLID and DLID, respectively, stored in the QP or EE Context.

9.6.1.5.2 IBA Unreliable Multicast Checks

C9-58: A packet is declared to be an IBA unreliable multicast packet if the destination QP is 0xFFFFFF. To be considered valid, it must have the following three characteristics: The packet must contain a GRH, the DGID must be a valid multicast GID, and there must be at least one locally managed queue pair configured for multicast operation. If any of these conditions is not true, the packet is not a valid multicast packet and shall be dropped silently.

C9-59: The DGID shall be used to map the inbound packet to a particular locally managed QP.

multicast_check

- good: a multicast packet meets all the criteria cited above to be a valid multicast packet.
- bad: a multicast packet does not meet all the criteria cited above to be a valid multicast packet.

9.7 RELIABLE SERVICE

Reliable Service provides a guarantee that messages are delivered from a requester to a responder at most once, in order and without corruption. Key elements of the reliable service include a protection scheme to enable detection of corrupted data (CRC), an acknowledgment mechanism allowing the requester to ascertain that the message had been successfully delivered, a packet numbering mechanism to detect missing packets and to allow the requester to correlate responses with requests, and a timer to allow detection of dropped or missing acknowledgment messages.

This section addresses the acknowledgment and packet sequence numbering mechanisms. The CRC mechanism for detecting packet corruption is not addressed here. Note that CRCs are checked at lower protocol layers and may result in packets being dropped before they are delivered to the transport layer. These dropped packets may eventually be detected at the transport layer as sequence errors.

- Characteristics of reliable service
  - Messages delivered at most once, in order and without corruption in the absence of unrecoverable errors.
  - Each message is acknowledged either explicitly or implicitly.
- Types of reliable service
  - Reliable Connection
  - Reliable datagram
- Reliability mechanisms
  - ACK / NAK protocol
  - Packet Sequence Numbers (PSN)
- Responder considers operation complete when it has:
  - Received a valid “Last” or “Only” OpCode in the BTH,
  - Received all packets comprising the message in proper PSN order,
  - Payload has been committed to the local fault zone (SENDS or RDMA WRITEs),
  - Response has been committed to the wire for RDMA READs or ATOMIC Operations,
  - Acknowledge packet for the last packet of the request has been committed to the wire (including the appropriate fields for RDMA READ response)

- Requester considers the operation complete when:
  - All packets of the response (for RDMA READ or ATOMIC Operation) have been received and committed to local memory,
  - Acknowledge message has been received and validated.

C9-60: Before it can consider a WQE completed, the requester must wait for the necessary response(s) to arrive. If the requester requires an explicit response such that it can complete a given WQE, then the requester shall be responsible to take the necessary steps to ensure that the needed response is forthcoming.

There are several mechanisms available to accomplish this such as:

5) Set the AckReq bit on the last packet of every message thus guaranteeing that the responder will generate the needed explicit response,
6) Set the AckReq bit on the last packet of the message for which an explicit response is desired,
7) If the AckReq bit was not set for the request for which an explicit response was desired, the requester can retry the request (with AckReq set) thus requiring the responder to return a response,
8) If the AckReq bit was not set for the request for which an explicit response was desired, the requester can send a NOP command (e.g. RDMA WRITE request with a length of zero) and set the AckReq bit. This strategy only works if the responder supports RDMA WRITES.

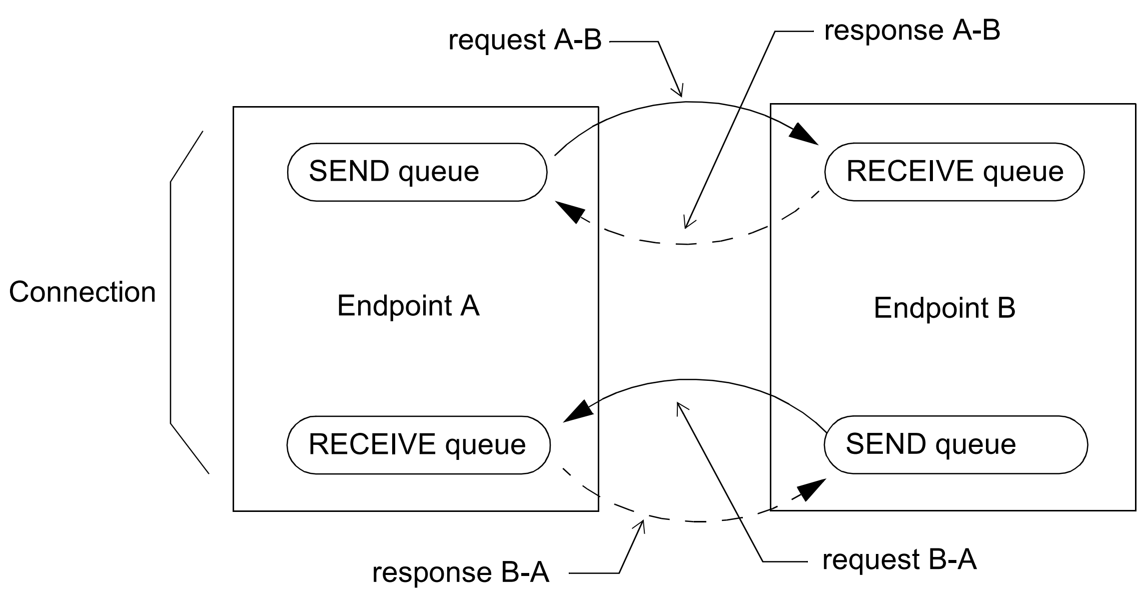
The choice of which of these, or other, strategies to use is implementation dependent.

9.7.1 Packet Sequence Numbers (PSN)

PSNs are transmitted within the Base Transport Header (BTH) for all packets. They are used to detect missing or out-of-order packets, and, for reliable services, to relate a response packet to a given request packet. Each IBA QP consists of a send queue and a receive queue; likewise, an EE Context has a send side and a receive side. There is a relationship between the PSN on a requester's send queue and the PSN on the responder's corresponding receive queue. Thus, each half of a QP (or EE Context) maintains an independent PSN; there is no relationship between the PSNs used on the Send queue and Receive queue of a given queue pair, or between different connection. This is illustrated in the figure below.

There are two abnormal conditions that must be detected and resolved at the responder to ensure reliable operation. The two conditions are duplicate packets and invalid packets.

1) Duplicate Packet. A duplicate packet may be recognized by the responder if the requester injects a request packet into the fabric more



Request A-B and Response A-B (for reliable service) are related by PSN A-B.
PSN A-B has no relationship to PSN B-A.

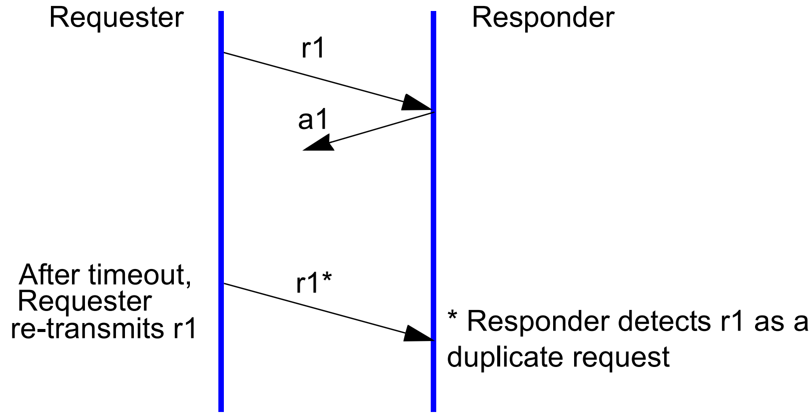
Figure 81 Send-Receive Queues Related by PSN

than once. This occurs when the requester detects a condition for which the prescribed recovery mechanism is to retry the operation.

There are two primary causes of a timeout condition that may cause the requester to inject a given request packet into the fabric more than once:

- A response is late in arriving at the requester either because a response packet is lost or delayed in the fabric as shown in Figure 82 below, or because the responder experienced a delay in generating a response, or
- A request packet may be lost or delayed in reaching the responder as shown in Figure 84.

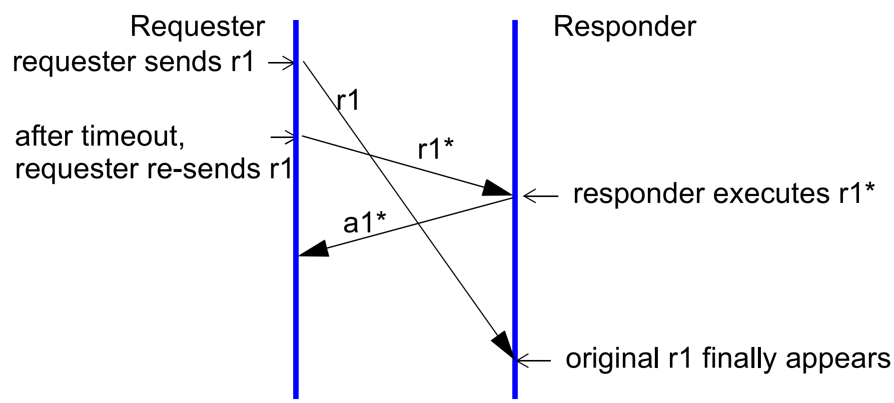
Regardless of the cause, the responder must be able to determine if an inbound request is a duplicate request that had been previously executed (or not) and respond appropriately.



'r' is a request packet,
'a' is an acknowledge packet

Figure 82 Duplicate Request Packets

In the previous figure, the response has been lost or delayed in the fabric causing the requester to detect a timeout condition and retransmit the request. The responder interprets the second arrival of the request packet as a duplicate request.

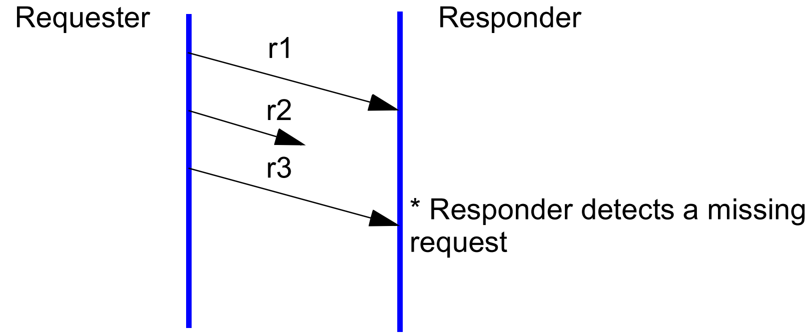


'r' is a request packet,
'a' is an acknowledge packet

Figure 83 Ghost Request Packet

A duplicate packet may also be detected by the responder due to a "ghost" request packet. This occurs when a request packet is delayed in the fabric long enough to cause a timeout to occur at the requester. The requester re-sends the original request packet to which the responder generates the proper acknowledge message. Sometime later, the original (delayed) packet arrives at the responder which interprets the late arriving packet as a duplicate request. This may occur, for example, during automatic path migration.

2) Invalid Request Packet Sequence. This condition occurs when the responder believes that one or more request packets have been lost in the fabric. This is illustrated in the following figure.



'r' is a request packet,
'a' is an acknowledge packet

Figure 84 Lost Request Packet(s)

The distinction between an invalid packet and a duplicate packet is important since the requester's actions and the responder's actions are different for the two cases.

These two conditions must be detected both by the responder (for request packets), or by the requester (for response packets on reliable services).

In addition to duplicate packets and invalid packets, there is a third condition, called a Stale Packet ("TIME WAIT packet"). If a connection to a responder is torn down and a new connection is established while packets are in flight, a packet from the old (stale) connection may arrive at the responder. The responder, in turn, may interpret this stale incoming packet as a valid packet, when in fact it is a remnant of a previous connection. There are no transport layer mechanisms to guard against this condition; it is the responsibility of connection management to avoid re-using QPs until there is no possibility that a stale packet could arrive at the responder. This is done by placing the requester and responder QPs in a "Time Wait" state long enough to ensure that any stale packets left in the fabric have expired before re-using those QPs.

Duplicate packets are distinguished from invalid packets by the 24-bit PSN field which is carried in the base transport header, and allows room for uniquely naming up to 16,777,216 packets.

C9-61: In order to make it possible for the responder to distinguish duplicate packets from out of order packets, a given send queue shall have a series of PSNs no greater than 8,388,608 outstanding at any given time. Therefore, a send queue shall have no more than 8,388,608 packets outstanding at any given time. This includes the sum of all SEND request packets plus all RDMA WRITE request packets plus all ATOMIC Operation request packets plus all expected RDMA READ response packets.

Thus, the PSN space (consisting of a range of 16,777,216 PSNs) is divided into two regions, each occupying a range of 8,388,608 PSNs, called the valid region and the invalid region. This is illustrated in Figure 85.

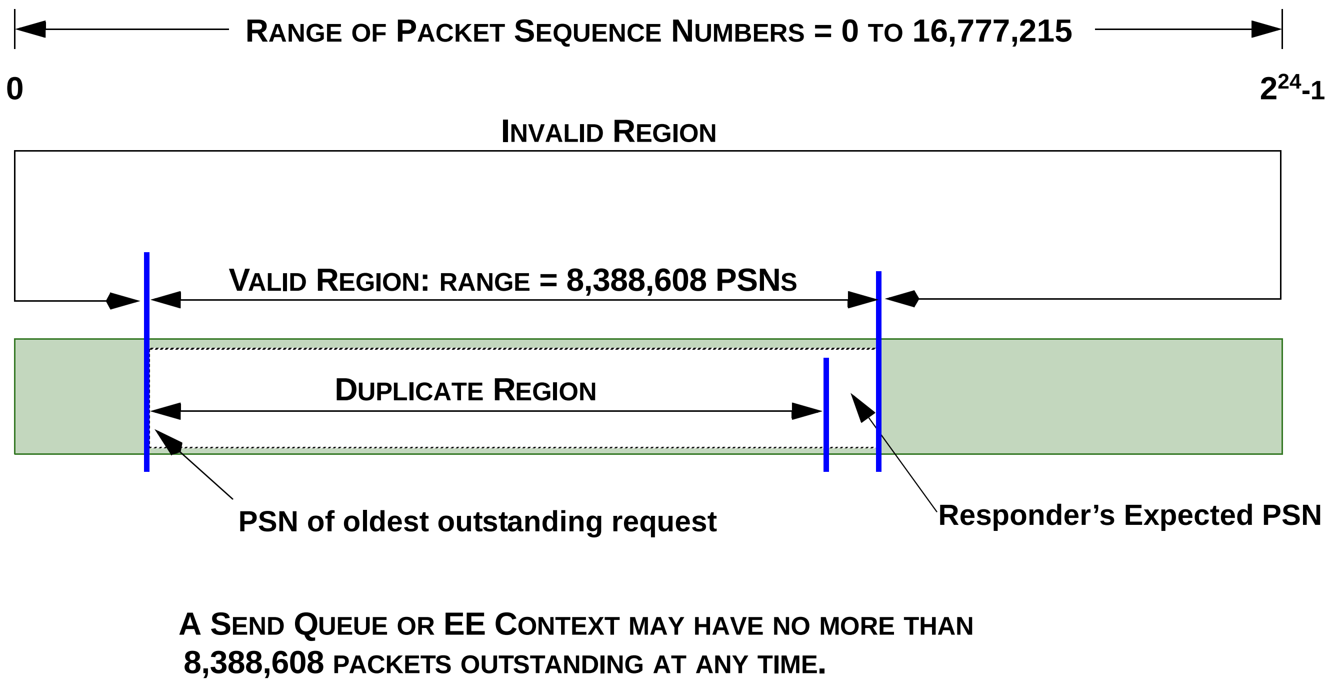


Figure 85 Valid and Invalid PSN Regions

The responder further subdivides the valid region into an Expected PSN and a Duplicate region. The responder's expected PSN (ePSN) is defined in Section 9.7.4.1.2 Responder - PSN Verification on page 264, and is simply described as the PSN that the responder expects to find in the next new request packet to be received. The duplicate region is therefore defined to be the entire valid region, except for the single expected PSN. Simply put, a duplicate PSN is a PSN which the responder has seen and executed previously and which falls within the valid region.

9.7.1.1 PSN MODEL FOR RELIABLE SERVICE

C9-62: For an HCA requester using Reliable Connection service, the requester shall insert a PSN in each packet of each request it generates. When responding to the request, the responder shall insert a PSN in each packet of each response it generates.

o9-29: If a TCA implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the requester shall insert a PSN in each packet of each request it generates. When responding to the request, the responder shall insert a PSN in each packet of each response it generates.

Except for the special case of RDMA READ responses, there is a 1:1 relationship between the PSN in a request packet and the PSN in the corresponding response packet.

In the general PSN model, the requester calculates the PSN of the next request packet to be generated. This calculated PSN is called the Next PSN. At the time that the requester generates a new request packet, the "Next PSN" is copied into the BTH and thus becomes the current PSN. The requester then calculates a new "Next PSN".

In order to detect missing or out of order packets, the responder also calculates the PSN it expects to find in the next inbound request packet. This is called the Expected PSN.

Conversely, when generating responses, the responder calculates the Response PSN to relate the response to a given request. However, due to acknowledge coalescing as described in 9.7.5.1.2 Coalesced Acknowledge Messages on page 275, the requester cannot necessarily predict which one of a range of PSNs may appear in the next response packet. Therefore, the requester must be prepared to accept any one of a range of Response PSNs. The range is bounded by the PSN of the oldest unacknowledged request packet and the expected response PSN of the most recently sent request packet. The requester evaluates the PSN of an inbound response packet to ensure that it falls between these two extremes. This general model is illustrated below.

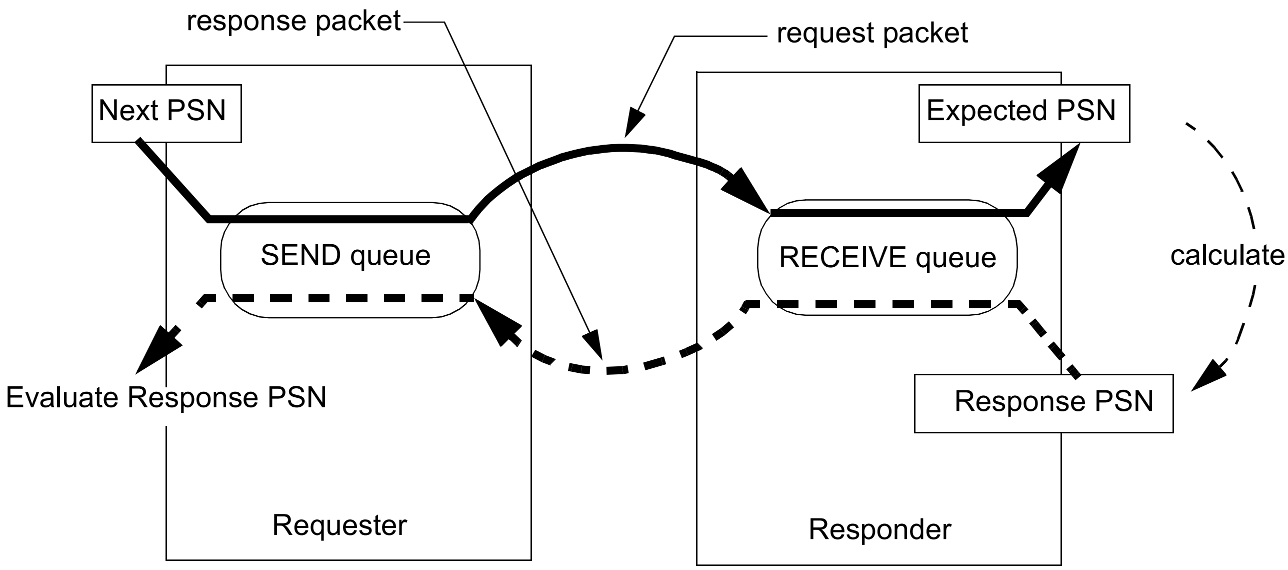


Figure 86 Request/Response PSNs

In the following sections only rarely is it not obvious from the context to which of the four PSNs the text is referring. Thus, it is common practice to refer to “PSN”, or “expected PSN” or some other variant. In the cases where the context is not clear, the above expressions are used for clarity.

9.7.2 ACK/NAK PROTOCOL

The ACK/NAK protocol, along with packet sequence numbers, is a fundamental component of reliable service, and applies to both reliable connected service and reliable datagram service. This and the following sections describe the protocol, provide a set of rules governing generation of ACK and NAK responses, specify the ACK and NAK codes and specify the requester’s required responses when it receives either an ACK or a NAK response.

The purpose of the ACK/NAK protocol is to allow the requester to ascertain deterministically if the responder correctly received the request packet. There are also mechanisms provided to ensure that a complete message was received correctly. This is accomplished through a combination of the packet sequence number and packet OpCodes (first/middle/last/only packet indications).

Since a response packet(s) can get lost in the fabric, the ACK/NAK protocol requires a requester to implement a timer to detect lost response packets. The transport timer is also described in this section.

The word “acknowledge” is used consistently throughout this section to mean either a negative (NAK) or a positive (ACK) acknowledgment. The generic term “response” is used to describe the acknowledgment returned by the responder to the requester. A response is carried in one or more acknowledge packets and may comprise, depending on the original request message, an ACK packet, a NAK packet, a RDMA READ response or an ATOMIC Operation response.

The following is a summary of the rules governing the ACK/NAK protocol:

- Each request packet received on a reliable service shall be acknowledged.
- Each RDMA READ request requires an explicit response. A RDMA READ response, with a properly formed ACK Extended Transport Header (AETH) is considered a valid response. The ACK Extended Transport Header appears in the first packet and last packet (or only packet) of a RDMA READ response. The details are covered below in Section 9.7.5.1.9 RDMA READ Responses on page 289
- Each ATOMIC Request requires an explicit response. An acknowledge packet, with a properly formed ACK Extended Transport Header (AETH) and an ATOMIC ACK Extended Transport Header (AtomicAckETH) is considered to be a valid response.
- Acknowledges may be coalesced; that is, a single acknowledge packet can serve as acknowledgment for one or more previous request packets.
- Acknowledge packets shall be returned in the PSN order in which the original request packet was received, including RDMA READ responses.
- A RDMA READ response consists of one or more packets; all other responses consist of exactly one packet.

C9-63: For an HCA responder using Reliable Connection service, the responder shall behave as follows. A responder shall acknowledge each request packet received. A responder shall generate an explicit response for each RDMA READ request received. A responder shall generate an explicit response for each ATOMIC Request received. A responder shall generate response packets in the PSN order in which the original request packets were received, including RDMA READ responses.

o9-30: If a TCA responder implements Reliable Connection service, or if a CA responder implements Reliable Datagram service, the responder shall behave as follows. A responder shall acknowledge each request packet received. A responder shall generate an explicit response for each RDMA READ request received. A responder shall generate an explicit response for each ATOMIC Request received. A responder shall generate response packets in the PSN order in which the original request packets were received, including RDMA READ responses.

9.7.3 Requester: Generating Request Packets

This section specifies the requirements placed on a requester as it generates request packets.

9.7.3.1 Requester Side - Generating PSN

C9-64: For Reliable Connection service in an HCA, the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

o9-31: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

During connection establishment, the transport layer's client programs the next PSN to any value between zero and 16,777,215. For proper operation, the initial expected PSN value on the responder side must be loaded with the same value.

C9-65: For Reliable Connection service in an HCA, the initial PSN, as programmed by the transport layer's client, is the PSN that shall appear in the first request packet generated by the requester.

o9-32: If Reliable Datagram service is implemented in CA, or if Reliable Connection Service is implemented in a TCA, then the initial PSN, as programmed by the transport layer's client, is the PSN that shall appear in the first request packet generated by the requester.

Thereafter, the requester calculates the next PSN. The calculation depends on the operation being performed (SEND, RDMA READ, etc.) and the size of the data payload.

With one exception, the requester shall increment the current PSN value by one for each request packet it generates. The single exception is for any request packet immediately following a RDMA READ request message. In this case, the request packet immediately following the RDMA READ request message shall have a PSN that is one greater than the PSN of the last expected RDMA READ response packet. In this way, the requester leaves a “hole” in the PSN sequence of the request packets, such that all response packets will have monotonically increasing PSNs.

Thus, for RDMA READ Requests:

Let curr_PSN = PSN of a RDMA READ Request

Let next_PSN = PSN of the request following a RDMA READ Request

Let n = the number of expected RDMA READ response packets

Then next_PSN = (curr_PSN + n) modulo 2^{24}

Table 39 Requester's Calculation of Next PSN

Current Request Packet	PSN for Next Request Packet
SEND, RDMA WRITE, ATOMIC Operation	current PSN + 1 (modulo 2^{24})
RDMA READ	current PSN + (number of expected RDMA READ response packets) (modulo 2^{24})

Since the requester knows both the total length of the requested RDMA READ data and the PMTU between the requester and the responder, and since there is a requirement that each response packet (except a last or only packet) be filled to the full PMTU size, the requester can calculate the total number of expected response packets and thus calculate the PSN of the request immediately following the RDMA READ request.

C9-66: For an HCA requester using Reliable Connection service, the requester shall behave as follows. For each request packet other than the packet immediately following an RDMA READ request message, the requester shall increment the next PSN value by one modulo 2^{24}. For any request packet immediately following a RDMA READ request message, the packet shall have a PSN that is one greater (modulo 2^{24}) than the PSN of the last expected RDMA READ response packet.

o9-33: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the requester shall behave as follows. For each request packet other than the packet immediately following an RDMA READ request message, the requester shall increment the next PSN value by one modulo 2^{24}. For any request packet immediately following a RDMA READ request message, the packet shall have a PSN that is one greater (modulo 2^{24}) than the PSN of the last expected RDMA READ response packet.

9.7.3.2 REQUESTER - SPECIAL RULES FOR RELIABLE DATAGRAM

9.7.3.2.1 RDD CHECKING

For reliable datagram service, any given send queue is associated with an EE Context by a Reliable Datagram Domain (RDD). Each send queue and EE Context has a single RDD associated with it. Before sending a request, the EE context must check the RDD of the currently active send queue. If the send queue's RDD does not match the EE Context's RDD, the current message transfer is terminated and a timeout condition is indicated to the send queue.

o9-34: For each request, the requester must confirm that the RDD of the currently active send queue matches the RDD of the selected EE context.

o9-35: If the send queue's RDD does not match the EE Context's RDD, the current message transfer is terminated and a Local ACK timeout condition must be indicated to the send queue.

9.7.3.2.2 RESYNC GENERATION

Under some conditions, a requester's EE Context is required to generate a special form of a request packet called a RESYNC request. This occurs when the requester EE Context elects to discontinue (abort) the current request message. The process of aborting the current request message and generating the subsequent RESYNC request is described in 9.7.8 Reliable Datagram on page 323 in the subsection dealing with handling QP errors.

The RESYNC request has the special property that it forces the responder to re-initialize its expected PSN to a value defined by the requester side EE Context.

The following paragraph governs the PSN that the requester is required to use when aborting the current message and generating the RESYNC request. It also governs the PSN it should expect in an acknowledgement received in response to the RESYNC request. These PSNs are identified in Figure 86 Request/Response PSNs on page 254 as the Next PSN and the Response PSN.

o9-35.a1: For a CA which supports Reliable Datagram service, when aborting the current request message and generating a RESYNC request, the requester side EE Context shall set the PSN of the RESYNC request to an increment of one greater (Modulo 2^{24}) than the largest PSN (Modulo 2^{24}) used in transmitting the current request. In addition, the requester shall reset the PSN it expects in a response to the same value. "The largest PSN used in transmitting the current request" means;

1) The (logically) largest PSN (Modulo 2^{24}) assigned to any packet of the request message including any retries of the request message, or
2) the PSN of the last expected RDMA READ response, if the request was an RDMA READ request.

For example, if the current request being transmitted consists of three packets with PSNs numbered 4, 5 and 6, then the PSN of the RESYNC request would be set to 7.

Or, consider the case where the requester is sending a large request message consisting of many packets. Even if the responder returns a NAK early in the transfer but the requester has sent subsequent packets, the PSN of the RESYNC request must be one greater (Modulo 2^{24}) than the PSN of any request packet sent, regardless of when the NAK packet was generated or its PSN.

In another example, if the current request being transmitted is an RDMA READ request with a PSN of 24, and to which the requester expects to receive 5 response packets, then the PSN of the RESYNC request would be set to 29, which is one greater than the PSN of the last expected RDMA READ response packet.

If a requester performs one or more retries, due to timeouts or other reasons, the PSN of the RESYNC request must be one greater (Modulo 2^{24}) than the PSN of any request packet sent, or response packet expected, for all of the previous attempts.

o9-35.a2: For a CA which supports Reliable Datagram service, the requester shall insert in the RESYNC request source and destination QPns which are identical to the source and destination QPs from the current request (which is being aborted).

9.7.3.3 Requester - Generating Opcodes

The opcodes generated by a requester must fit into a schedule of opcodes as shown below.

C9-67: A requester must generate packet opcodes which fit within the schedule of valid OpCode sequences as shown in Table 40 Schedule of Valid OpCode Sequences on page 260.

Table 40 Schedule of Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

C9-68: When generating a request packet, the BTH:Opcode shall be as specified in Table 35 OpCode field on page 207.

9.7.3.4 REQUESTER - GENERATING PAYLOADS

The requester shall generate payload lengths as a function of the opcode as follows:

C9-69: If the OpCode specifies a "first" or "middle" packet, then the packet payload length must be a full PMTU size.

C9-70: If the OpCode specifies a "only" packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

C9-71: If the OpCode specifies a "last" packet, then the packet payload length must be between one and PMTU bytes in size.

C9-72: For an HCA, if the OpCode specifies an RDMA WRITE request, the length specified in the DMALen field of the RETH shall be no less than zero, and no greater than 2^{31} bytes.

o9-36: If RDMA WRITE is implemented in a TCA and the OpCode specifies an RDMA WRITE request, the length specified in the DMALen field of the RETH shall be no less than zero, and no greater than 2^{31} bytes.

9.7.4 RESPONDER: RECEIVING INBOUND REQUEST PACKETS

This section describes the process used by a responder when it receives an inbound request packet.

9.7.4.1 RESPONDER - INBOUND PACKET VALIDATION

C9-73: For Reliable Connection service in an HCA, inbound request packets shall be validated as shown in Figure 87 on page 262.

o9-36.a1: If Reliable Connection service is implemented in a TCA, inbound request packets shall be validated as shown in Figure 87 on page 262.

o9-37: If Reliable Datagram service is implemented in a CA, inbound request packets shall be validated as shown in Figure 88 on page 263.

The following sections describe each of the validation checks and the responder's behavior / response.

9.7.4.1.1 RESPONDER - SPECIAL RULES FOR RELIABLE DATAGRAM CHECKING

o9-38: For RD within a HCA, when an inbound packet arrives, the receive queue must test its own RDD value against that of the EE Context over which the inbound packet arrived. If they do not match, the receive queue must drop the packet and schedule a NAK-Invalid RD Request. The P_Key and PSN to be used for returning the NAK shall be supplied by the EE Context.

o9-38.a1: If Reliable Datagram service is implemented in a CA, a responder shall do the following when an inbound packet arrives:

1) If the responder receives a new request packet (i.e. PSN = expected PSN) with an opcode of "middle" or "last" (i.e., the responder has previously received a new request packet with an opcode of "first" and has not yet received a new request packet with an opcode of "last"), the responder shall validate that the source and destination QPns contained in the new request packet exactly match those of the most recently received "first" packet. If the source and destination QPns of the new "middle" or "last" request packet do not match those of the most recently received "first" request, the responder shall ignore the packet.
2) If the responder receives a duplicate request packet (i.e., the PSN is in the duplicate region), the responder shall validate that the source and destination QPns in the duplicate request packet exactly match those of the most recently received new request. If the source and destination QPns do not match those of the most recently received new request, the responder shall ignore the packet.

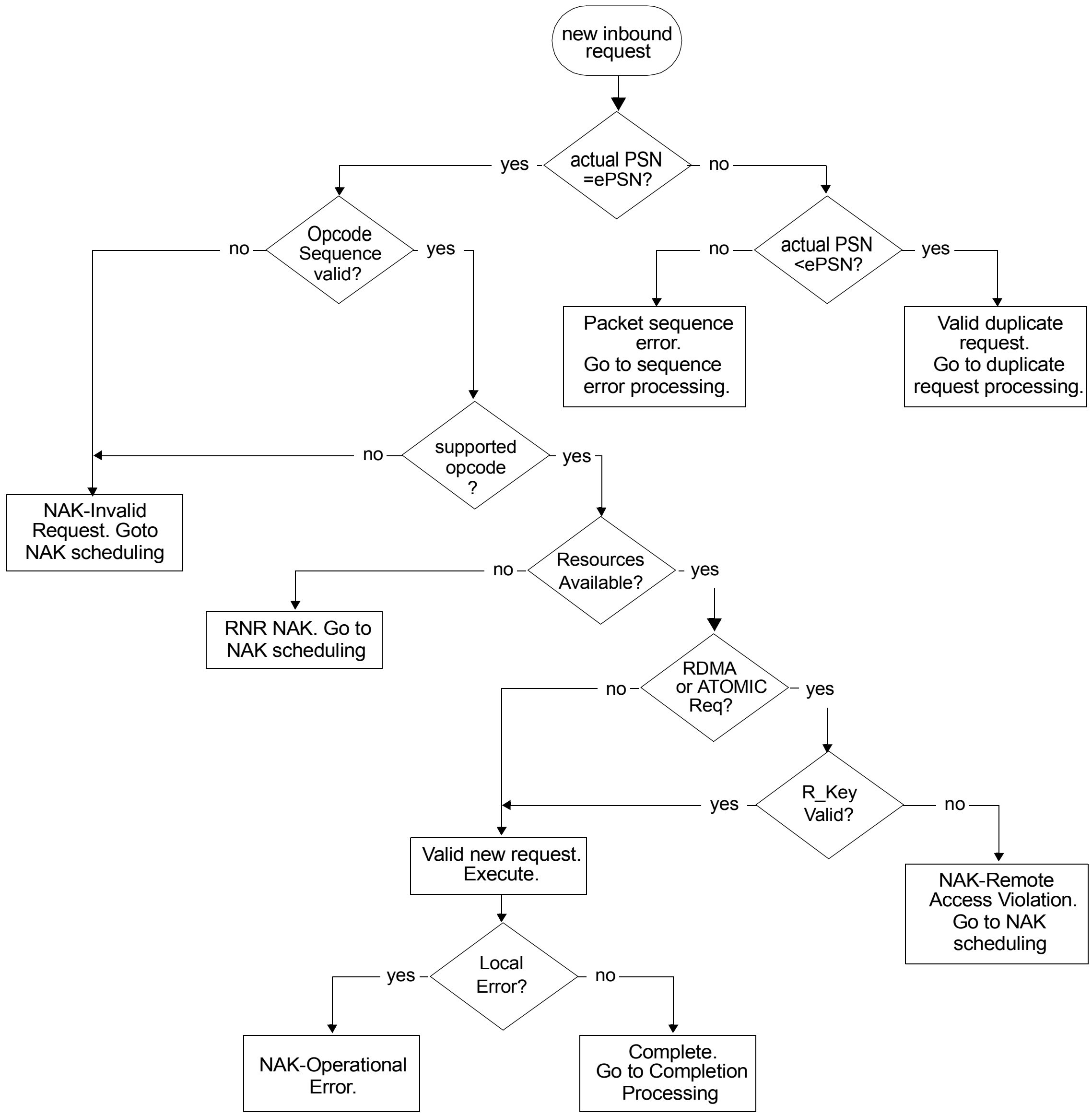


Figure 87 Inbound Request Packet Validation, RC mode

OM10528

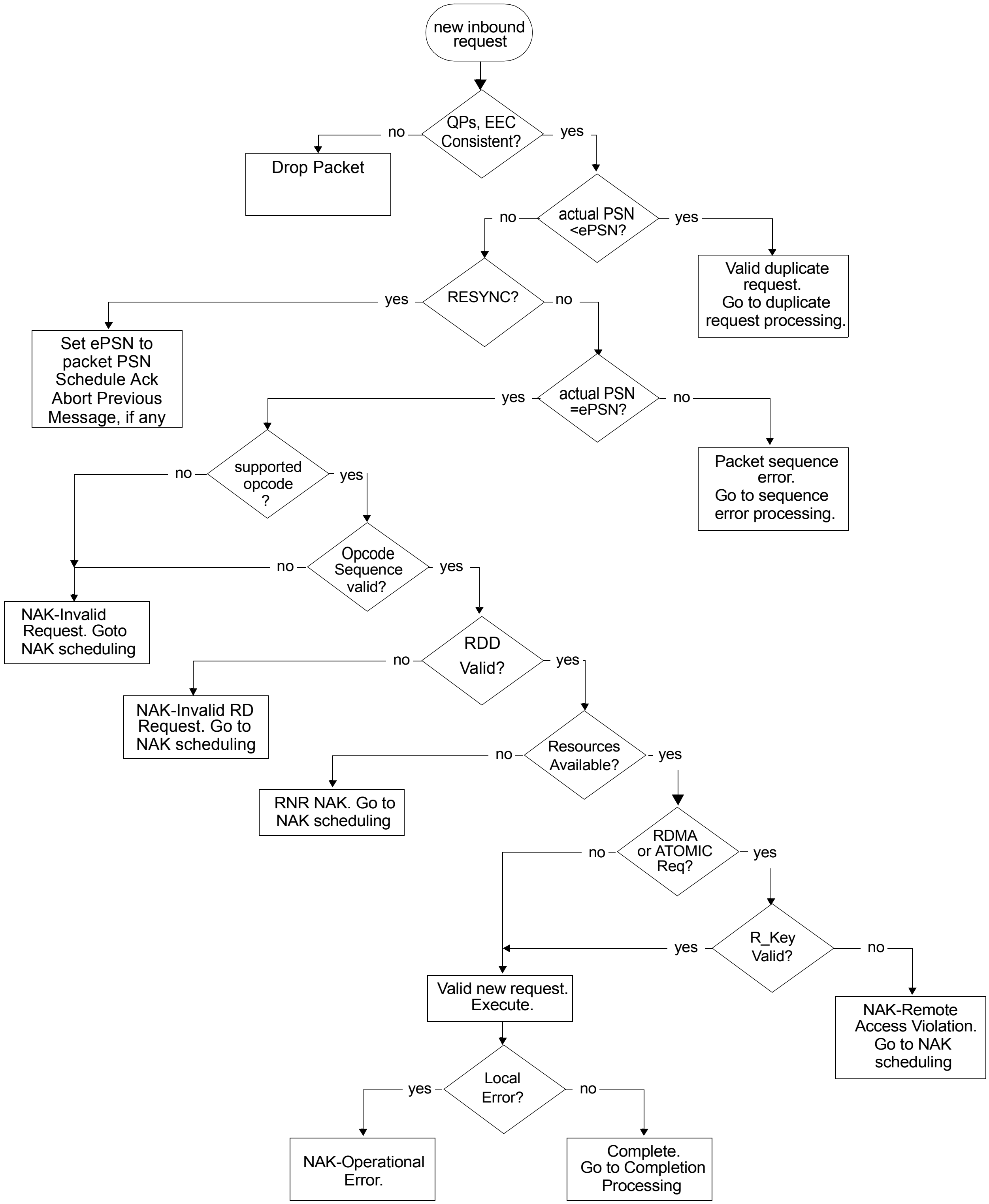


Figure 88 Inbound Request Packet Validation, RD mode

o9-38.a2: If Reliable Datagram service is implemented in a CA, when an inbound RESYNC request arrives, the responder shall do the following:

1) If the responder receives a RESYNC request, it shall accept the request regardless of the value of the source and destination QPns in the RESYNC request.
2) If the responder receives a RESYNC request, it shall accept that request regardless of the state of the opcode sequence (“first”, “middle”, etc.) of the currently executing request (i.e. the most recently received new request).
3) If the PSN of the RESYNC request is equal to or logically greater than the responder’s expected PSN (i.e. the RESYNC request is a new request), the responder shall:
    a) Set its expected PSN equal to the PSN of the RESYNC request plus one (Modulo 2^{24}),
    b) Use the PSN of the RESYNC request as the PSN of the response,
    c) Abort any processing associated with the currently executing request if the currently executing request is incomplete (i.e. a Request packet with an opcode of “last” or “only” has not yet arrived). If such a currently executing request was a SEND RDMA WRITE with Immediate (and thus would have consumed a receive WQE), the partially consumed receive WQE shall be completed in error.
4) If the PSN of the RESYNC request is logically less than the responder’s expected PSN, the responder shall treat the RESYNC request as a duplicate request and thus shall not change its expected PSN. The responder shall use the PSN of the duplicate RESYNC request as the PSN of the response.

9.7.4.1.2 RESPONDER - PSN VERIFICATION

C9-74: For Reliable Connection service in an HCA responder, and before executing the inbound request, the responder shall check the PSN by comparing the inbound BTH:PSN to the responder’s expected PSN. The PSN shall be checked by the responder’s receive queue.

o9-39: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and before executing the inbound request, the responder shall check the PSN by comparing the inbound BTH:PSN to the responder’s expected PSN. The PSN shall be checked by the responder’s receive queue.

For reliable datagram service, the PSN is checked by the responder’s EE Context.

To a large extent, the responder's behavior in responding to a request is based on an interpretation of the incoming PSN.

Logically, a receive queue or EE Context maintains an expected PSN (ePSN). This is the PSN that the responder expects to find in the BTH of the next new request packet it receives. The rules that the responder uses to calculate its next expected PSN are the same as those used by the requester when it calculates the PSN value to insert in its next request packet.

C9-75: For Reliable Connection service in an HCA responder, a responder shall use the rules given in 9.7.3.1 Requester Side - Generating PSN on page 256 to calculate its expected PSN.

o9-40: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, a responder shall use the rules given in 9.7.3.1 Requester Side - Generating PSN on page 256 to calculate its expected PSN.

The responder's expected PSN is initialized at connection establishment time by the Connection Manager to any value between zero and 16,777,215. For proper operation, this initial value must match the initial next PSN value as loaded on the requester.

The initial expected PSN can only be set by the client when the queue is in the Initialized state. Attempts by the client to set the PSN when it is in any other state may be ignored by the transport layer.

C9-76: For Reliable Connection service in an HCA responder, the HCA shall update its expected PSN only when the receive queue is in a state such that it is properly conditioned to receive request packets. For example, the transport does not modify the expected PSN when the queue pair is in the Initialized state.

o9-41: If Reliable Connection service is implemented in a TCA, the responder shall update its expected PSN only when the Receive Queue is in a state such that it is properly conditioned to receive request packets. For example, the transport does not modify the expected PSN when the queue pair is in the Initialized state.

o9-42: If Reliable Datagram service is implemented in a CA, the responder shall update its expected PSN only when the EE Context is in a state such that it is properly conditioned to receive request packets.

When compared to its expected PSN, the actual PSN of an inbound request message may fall into one of three regions; it may be exactly equal to the responder's expected PSN, it may be logically "less" than the responder's expected PSN and thus fall into the duplicate region as shown in Figure 85, or it may fall outside both the valid region and the expected PSN "region", and thus be invalid.

Expected (new) Request: An inbound request packet received with a PSN that exactly matches the responder's expected PSN is a new request packet.

C9-77: For Reliable Connection service in an HCA responder, a new request packet shall be validated normally and executed according to the rules governing order of execution. Once the request has been executed, a response shall be scheduled as specified in 9.7.5 Responder: Generating Responses on page 273.

o9-43: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, a new request packet shall be validated normally and executed according to the rules governing order of execution. Once the request has been executed, a response shall be scheduled as specified in 9.7.5 Responder: Generating Responses on page 273.

Note that it is not required to return a discrete acknowledge packet for each inbound request packet.

Once a packet with a valid expected PSN has been received, the responder advances its expected PSN by calculating the new expected PSN, and slides the valid region window up to reflect the new range of valid PSNs.

Valid Duplicate Request: A PSN that falls within the valid region, but is not the expected PSN, is a valid duplicate request packet.

C9-78: For Reliable Connection service in an HCA responder, the responder shall respond to valid duplicate requests as specified in 9.7.5.1.4 Acknowledging Duplicate Requests on page 279.

o9-44: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then the responder shall respond to valid duplicate requests as specified in 9.7.5.1.4 Acknowledging Duplicate Requests on page 279.

Table 41 summarizes those actions.

Table 41 Summary: Responder Actions for Duplicate PSNs

Duplicate Request Message	Responder Action
SEND or RDMA WRITE or RESYNC	Schedule acknowledge packet
RDMA READ	Re-execute request, schedule response
ATOMIC Operation	Do not re-execute request, after validating the request, return the saved results from the referenced ATOMIC Operation request.

Invalid Request: A packet with an actual received PSN outside the valid region and not in the expected “regions” is an invalid request. An invalid PSN value is generally an indication that one or more request packets have been lost in the fabric.

The responder’s detailed behavior in response to an invalid request request packet is as follows:

- The errant request packet is not executed.
- Any request packets received prior to the errant request must be executed and completed before the NAK-Sequence Error is issued since it acts as an implicit ACK for prior outstanding SEND or RDMA WRITE requests, and as an implicit NAK for outstanding RDMA READ or ATOMIC Operation requests.
- Return a NAK-Sequence error to the requester.
- The responder does not update its expected PSN.

C9-79: For Reliable Connection service in an HCA responder, when the actual PSN of an inbound request message is outside the valid region (Invalid Request), a NAK-Sequence Error shall be returned by the responder. Any request packets received prior to the errant request must be executed and completed before the NAK-Sequence Error is issued.

o9-45: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and if the actual PSN of an inbound request message is outside the valid region (Invalid Request), then a NAK-Sequence Error shall be returned by the responder. Any request packets received prior to the errant request must be executed and completed before the NAK-Sequence Error is issued.

The responder resumes waiting for a valid inbound request packet that has a PSN equal to its expected PSN or within its valid region. If, while waiting for a valid new request, the responder receives any subsequent invalid request packets, those packets are simply dropped silently; no NAK is returned.

C9-80: For Reliable Connection service in an HCA responder, after generating a NAK-Sequence Error, the responder shall not generate an ACK or NAK until it receives either a valid new request, or a valid duplicate request.

o9-46: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then after generating a NAK-Sequence Error, the responder shall not generate an ACK or NAK until it receives either a valid new request, or a valid duplicate request.

There is no requirement that the queue be stopped or for a connected transport service that the connection be torn down.

9.7.4.1.3 RESPONDER - OPCODE SEQUENCE CHECK

A request packet must fit within a schedule of valid OpCode sequences. For Reliable Connected and Reliable Datagram services the responder shall check the sequence of packet OpCodes comprising the request message as follows:

1) If this is the first packet following establishment of the connection, then the packet OpCode must indicate either “first” or “only”.
2) If the last valid packet received had an OpCode indicating “first”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (Send, RDMA, ATOMIC Operation). It is an error if the current OpCode indicates “first” or “only”, since that implies that the last packet of the previous message was missed.
3) If the last valid packet received had an OpCode indicating “middle”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (Send, RDMA, ATOMIC Operation). It is an error if the current OpCode indicates “first” or “only” packet since that implies that the last packet of the previous message was missed.
4) If the last valid packet received had an OpCode indicating “last”, then the current OpCode must indicate either “first” or “only”. It is an error if the current OpCode indicates either “middle” or “last”, since that implies that the first packet of the message was missed.
5) If the last valid packet received had an OpCode indicating “only”, then the current OpCode must indicate either “first” or “only”. It is an error if the current OpCode indicates either a middle packet or last packet since that implies that the first packet of the message was missed.

These rules are stated succinctly in the following table.

Table 42 Schedule of Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

C9-81: For an HCA responder using Reliable Connected service, the responder shall check that the sequence of packet OpCodes comprising the request message conforms to the schedule shown in Table 42 Schedule of Valid OpCode Sequences on page 269. If the responder detects an invalid opcode sequence, it shall return a NAK-Invalid Request to the requester.

o9-47: If a TCA responder implements Reliable Connected service, the responder shall check that the sequence of packet OpCodes comprising the request message conforms to the schedule shown in Table 42 Schedule of Valid OpCode Sequences on page 269. If the responder detects an invalid opcode sequence, it shall return a NAK-Invalid Request to the requester.

o9-48: If a CA responder implements Reliable Datagram service, the responder shall check that the sequence of packet OpCodes comprising the request message conforms to the schedule shown in Table 42 Schedule of Valid OpCode Sequences on page 269. If the responder detects an invalid opcode sequence, it shall return a NAK-Invalid Request to the requester.

The detailed behavior in the presence of an invalid OpCode sequence is specified in Section 9.9 Error detection and handling on page 362.

o9-49: This compliance statement is obsolete and has been removed.

9.7.4.1.4 RESPONDER OPCODE VALIDATION

C9-82: Before executing an inbound request, the responder shall validate the OpCode field of the BTH.

The OpCode is checked for the following characteristics:

- The requested function (Send, RDMA, ATOMIC) is supported by this receive queue,
- If the request is for an RDMA READ or an ATOMIC Operation, there are sufficient resources available to receive it.

As the packet was passed up to the transport layer, BTH OpCode field[7:5] was checked to ensure that the requested operation was for a reliable service. If it was not, then the packet was silently dropped. This check is specified in Section 9.6 Packet Transport Header Validation on page 236. Thus, before the packet arrives at the queue pair for validation according to the rules in this section, it is already known to be a request for a reliable service.

C9-83: For Reliable Connection service in an HCA responder, if the request is for a function which this receive queue does not support, then a NAK-Invalid Request shall be returned.

For example, if the queue pair is not configured to accept requests for RDMAs, but the request is for an RDMA WRITE, then a NAK-Invalid Request shall be returned.

o9-50: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, if the request is for a function which this receive queue does not support, then a NAK-Invalid Request shall be returned.

C9-84: For Reliable Connection service in an HCA responder, and the BTH OpCode field[4:0] specifies a Reliable Connection reserved opcode or a Reliable Datagram reserved opcode, a NAK-Invalid Request shall be returned.

o9-51: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and the BTH OpCode field[4:0] specifies a Reliable Connection reserved opcode or a Reliable Datagram reserved opcode, then a NAK-Invalid Request shall be returned.

C9-85: For Reliable Connection service in an HCA responder, if BTH OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA WRITE Middle), then the pad count bits shall be verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, a NAK-Invalid Request shall be returned.

o9-52: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, if BTH OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA

WRITE Middle), then the pad count bits shall be verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, a NAK-Invalid Request shall be returned.

C9-86: For Reliable Connection service in an HCA responder, if there are insufficient resources to receive a new RDMA READ or ATOMIC Operation request, then a NAK-Invalid Request shall be returned.

o9-53: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, and if there are insufficient resources to receive a new RDMA READ or ATOMIC Operation request, then a NAK-Invalid Request shall be returned.

The behavior for returning a NAK-Invalid Request is as follows:

- The errant request packet is not executed.
- Any request packets received prior to the errant request must be executed and completed before the NAK-Invalid Request is issued. This is important since the NAK effectively coalesces responses to earlier outstanding request and acts as an implicit response for prior outstanding SENDs, RDMA WRITEs, ATOMIC Operations or RDMA READ requests. See Section 9.7.5.1.2 Coalesced Acknowledge Messages on page 275 for details.
- NAK-Invalid Request is returned.
- The responder does not update its expected PSN.

C9-87: For Reliable Connection service in an HCA responder, any request packets received prior to a packet containing an invalid opcode must be executed and completed before a NAK-Invalid Request is issued by the responder.

o9-54: If Reliable Datagram service is implemented in a CA, or if Reliable Connection service is implemented in a TCA, then any request packets received prior to a packet containing an invalid opcode must be executed and completed before a NAK-Invalid Request is issued by the responder.

More detail on error behavior in the presence of an invalid request is given in Section 9.9.3 Responder Side Behavior on page 375.

9.7.4.1.5 RESPONDER R_KEY VALIDATION

A R_Key violation is caused by any or all of the following conditions for either a RDMA READ, RDMA WRITE, or ATOMIC Operation:

- The R_Key field of the RETH is invalid (for RDMA READ or WRITEs)
- The R_Key field of the AtomicETH is invalid (for ATOMIC Operations).

- The virtual address and length, or type of access specified, is outside the locally defined limits associated with the R_Key. For an RDMA WRITE request, the length check is conducted on a per packet basis, and is based on the LRH:PktLen field. For an RDMA READ request, the length check is based on the RETH:DMA Length field.

C9-88: For an HCA responder using Reliable Connection service, for each zero-length RDMA READ or WRITE request, the R_Key shall not be validated, even if the request includes Immediate data.

o9-55: If an HCA responder implements Reliable Datagram service, or if a TCA responder implements Reliable Connection and RDMA functionality, or if a TCA responder implements Reliable Datagram service and RDMA functionality, the responder shall behave as follows. For each zero-length RDMA READ or WRITE request, the R_Key shall not be validated, even if the request includes Immediate data.

C9-89: If the responder's receive queue detects a R_Key violation, a NAK-Remote Access Error shall be returned to the requester using the PSN of the errant request packet.

C9-90: Any request packets received prior to a packet containing an R_Key violation shall be executed and completed before a NAK-Remote Access Error is issued by the responder.

See 9.7.5.2.4 Remote Access Error on page 293 for additional details.

9.7.4.1.6 RESPONDER - LENGTH VALIDATION[1]

C9-91: The PktLen field of the LRH shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE. If the buffer defined by the receive WQE is insufficient to hold an inbound SEND request then a NAK-Invalid Request shall be returned.

C9-92: The length of the packet shall also be validated by comparing it to the OpCode as follows:

If the OpCode specifies a “first” or “middle” packet, then the packet payload length must be a full PMTU size.

If the OpCode specifies a “only” packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

If the OpCode specifies a “last” packet, then the packet payload length must be between one and PMTU bytes in size.

1. CAs are not required to validate the GRH packet length.

C9-93: If a packet is detected with an invalid length the request shall be an invalid request.

The responder's behavior in such a case is specified in Section 9.9.3 Responder Side Behavior on page 375.

In addition to checking the LRH:PktLen field, the DMA Length field of the RETH is checked as follows.

For an RDMA WRITE request, the responder may optionally check the DMA Length field in the RETH to ensure that it does not specify a transfer length of greater than 2^{31} bytes. It may also, at the end of the transfer, verify that the sum of the packet payloads equalled the DMALen field in the RETH. If the responder detects either of these conditions, it may treat the request as an invalid request.

C9-94: For an HCA validating an inbound RDMA READ request, the DMA Length field shall be checked. If the request is for greater than 2^{31} bytes, then a NAK-Invalid Request shall be returned.

o9-56: If a TCA implements RDMA operations, then for an inbound RDMA READ request, the DMA Length field shall be checked. If the request is for greater than 2^{31} bytes, then a NAK-Invalid Request shall be returned.

9.7.4.1.7 Responder Local Operation Validation

A valid inbound request may still fail to complete due to a failure that is local to the responder, e.g. local memory translation error while accessing local memory. All local responder errors are reported to the requester as NAK-Remote Operational Error. See 9.7.5.2.6 Remote Operational Error on page 294 for additional details.

9.7.5 Responder: Generating Responses

9.7.5.1 Responder Side Behavior

This section specifies the required behavior that a responder must follow when generating acknowledge messages.

9.7.5.1.1 Generating PSNs for Acknowledge Messages

As the responder generates a response to each request, it shall identify the request with which the response is associated by inserting a PSN in the BTH of the response.

This allows the requester to correlate response packets it receives with its request. This basic concept is illustrated below in Figure 89 Example: PSNs for Response Messages on page 274.

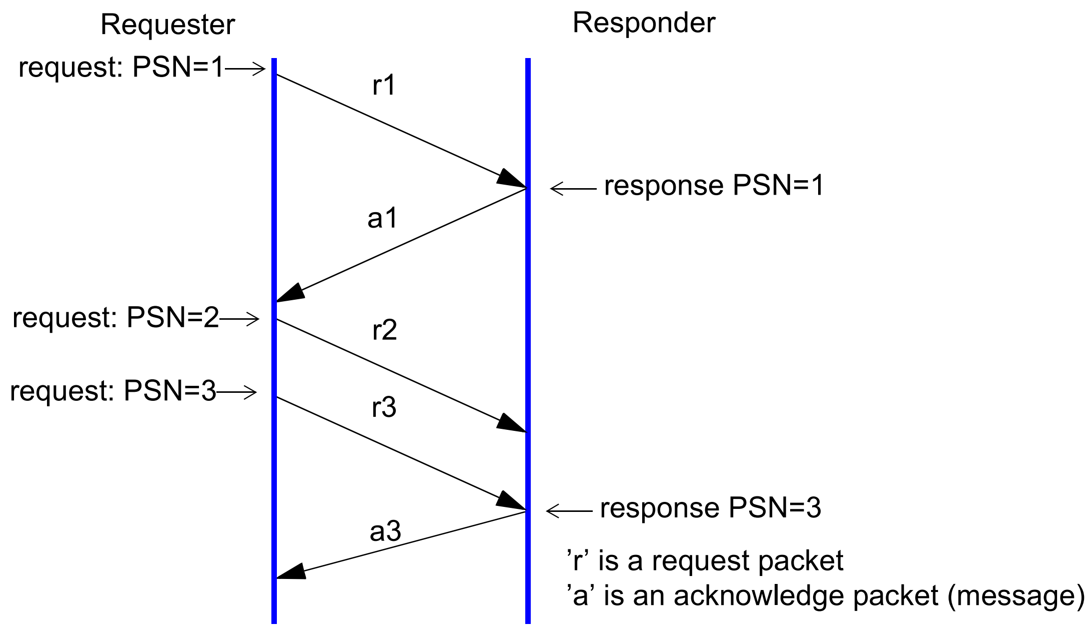


Figure 89 Example: PSNs for Response Messages

C9-95: For responses to SEND requests or RDMA WRITE requests the responder shall insert in the PSN field of the response the PSN of the most recent request packet being acknowledged.

Because of the rules for coalescing acknowledges (given in Section 9.7.5.1.2), the PSNs for consecutive response packets may not necessarily be sequential.

C9-96: For HCA responses to RDMA READ requests, the PSNs of the response packets must be sequential and monotonically increasing beginning with the PSN of the original RDMA READ request message.

o9-57: If a TCA implements RDMA READ functionality, then for each RDMA READ response the PSNs of the response packets must be sequential and monotonically increasing beginning with the PSN of the original RDMA READ request message.

o9-58: Since ATOMIC Operation requests require an explicit response, and since an ATOMIC Operation request is restricted to a single packet, the PSN of the response packet must be identical to the PSN of the request.

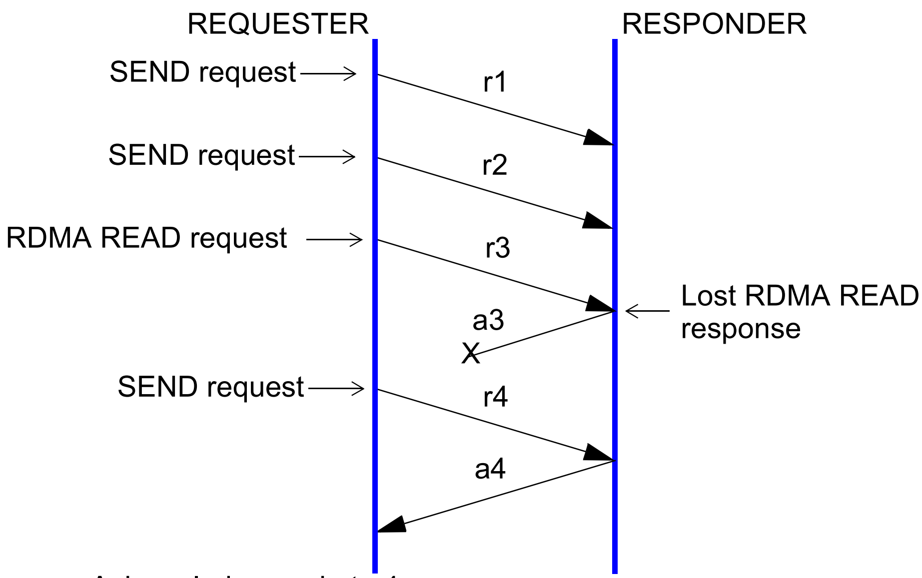
9.7.5.1.2 Coalesced Acknowledge Messages

It is not required that there be a unique, discrete response for each request packet. Instead, the responder may acknowledge several outstanding request packets with a single acknowledge packet. This is called acknowledge coalescing.

A given response packet acknowledges prior outstanding requests (i.e., those with earlier PSNs than the PSN contained in the BTH of the response packet) as follows:

1) If there is an outstanding RDMA READ or ATOMIC Operation request with a PSN earlier than the PSN in the BTH of the response packet, then the response packet implies a negative acknowledgment for the oldest such outstanding RDMA READ or ATOMIC Operation request. Any requests posted to the send queue subsequent to such an RDMA READ or ATOMIC Operation request are not acknowledged. This is illustrated in Figure 90 Requester Interpretation of Coalesced Acknowledges on page 276.
2) It implies a positive acknowledgment for all outstanding SEND or RDMA WRITE request packets with a PSN earlier than the PSN in the BTH of the response packet, unless such an outstanding SEND or RDMA WRITE request falls after an outstanding RDMA READ or ATOMIC Operation request as described above.
3) If the given response is an RDMA READ response message, it is the first (or only) packet of a RDMA READ response message that implicitly acknowledges prior outstanding requests.
4) The last (or only) packet of a RDMA READ response message explicitly acknowledges only the RDMA READ request.

These rules are illustrated in Figure 90.



Acknowledge packet a4:

1) implicitly acknowledges SEND requests r1 and r2,
2) implicitly NAKs RDMA READ request r3.
3) does not acknowledge SEND request r4.

Thus, acknowledges for requests r1 and r2 have been coalesced, and the requester must re-try requests r3 and r4.

Figure 90 Requester Interpretation of Coalesced Acknowledges

9.7.5.1.3 ACKNOWLEDGING RDMA READ REQUESTS

An RDMA READ response is different from a normal response in that it contains a data payload.

Every RDMA READ request message requires a discrete acknowledgment, called the RDMA READ response which consists of one or more packets.

C9-97: For an HCA, if an RDMA READ response contains more than one packet, the first and last packets must contain an AETH.

o9-59: In a TCA implementing RDMA, if an RDMA READ response contains more than one packet, the first and last packets must contain an AETH.

The AETH in the first packet implicitly acknowledges prior outstanding requests as specified in Section 9.7.5.1.2 Coalesced Acknowledge Messages on page 275. The AETH in the last packet acknowledges the RDMA READ request.

C9-98: For an HCA, if an RDMA READ response is itself a single packet, then that packet must contain an AETH.

o9-60: If a TCA implements RDMA functionality, and an RDMA READ response is itself a single packet, then that packet must contain an AETH.

The AETH contained in a single packet RDMA READ response serves to both implicitly acknowledge prior outstanding requests as well as to explicitly acknowledge the RDMA READ request itself.

C9-99: An HCA responder shall generate RDMA READ response packet payload lengths which are consistent with the opcode as follows:

1) A packet with an opcode of “RDMA READ response only” shall be zero to (PMTU) bytes long.
2) A packet with an opcode of “RDMA READ response first” or RDMA READ response middle” shall be exactly (PMTU) bytes long.
3) A packet with an opcode of “RDMA READ response last” shall be one to (PMTU) bytes long.
4) Zero length RDMA READ requests are permitted.
5) A response to a zero length RDMA READ request shall consist of a single packet with an opcode of “RDMA READ response only”.

o9-61: If a TCA implements RDMA functionality, it shall generate response packets with payload lengths as described in the previous compliance statement.

C9-100: If an HCA responder detects an error while in the process of returning a multi-packet RDMA READ response, it shall force a premature termination of the RDMA READ response by not transmitting any of the errant payload data and forcing the opcode of the packet on which the error occurred to “acknowledge” instead of an opcode of “RDMA READ response last”. The appropriate NAK code is inserted.

o9-62: If a TCA implements RDMA functionality, and detects an error while in the process of returning a multi-packet RDMA READ response, it shall force a premature termination of the RDMA READ response by not transmitting any of the errant payload data and forcing the opcode of the packet on which the error occurred to “acknowledge” instead of an opcode of “RDMA READ response last”. The appropriate NAK code is inserted.

Due to the relaxed ordering rules for RDMA READ Requests, the responder is permitted to begin executing one or more SEND or RDMA WRITE requests that arrive after the RDMA READ request.

C9-101: For an HCA, before executing any of the requests following the RDMA READ request, the header fields of the RDMA READ request must be validated. These requests must not be acknowledged until the outstanding RDMA READ responses have been sent.

o9-63: Before executing any of the requests following the RDMA READ request, the header fields of the RDMA READ request must be validated. These requests must not be acknowledged until the outstanding RDMA READ responses have been sent.

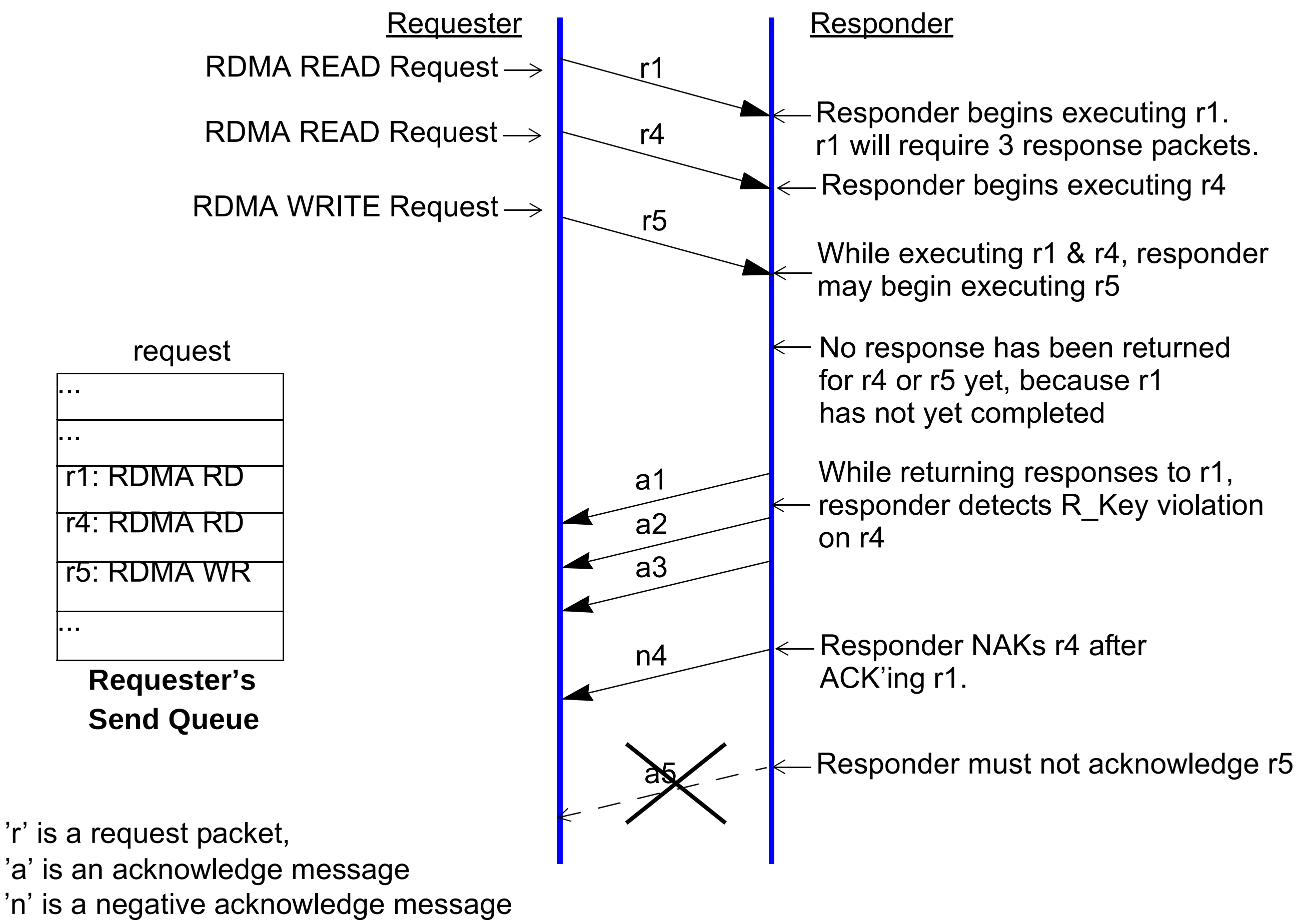


Figure 90 Relaxed Ordering Rules for RDMA READs

9.7.5.1.4 ACKNOWLEDGING DUPLICATE REQUESTS

After validating a duplicate request, the response to a duplicate request packet is as follows:

C9-102: After validating a duplicate request, if the duplicate packet is valid, the responder shall generate a response.

C9-103: Throughout the processing of the duplicate request, the responder shall not update its expected PSN; it remains set to the value it had prior to the arrival of the duplicate request. This is true even if the responder detects an error while in the process of generating the response to the duplicate request.

Following generation of the appropriate response (as described in the next paragraphs), the responder resumes waiting for a new inbound packet with a PSN matching its expected PSN.

It is possible that the responder will receive another duplicate request while waiting for a new inbound packet. This is perfectly valid, and should be treated as simply another duplicate request. Furthermore, since it is a duplicate request, there is no requirement that the next request received be in sequential PSN order with the first duplicate request. However, the responder is required to maintain the same ordering rules for generating responses to duplicate requests as are required for normal new requests.

C9-104: In particular, a duplicate RDMA READ or ATOMIC Operation request must be acknowledged with an explicit response prior to returning acknowledges for subsequent duplicate SEND or RDMA WRITE requests.

This is illustrated in Figure 91 Maintaining the Order of Responses to Duplicate Requests on page 281.

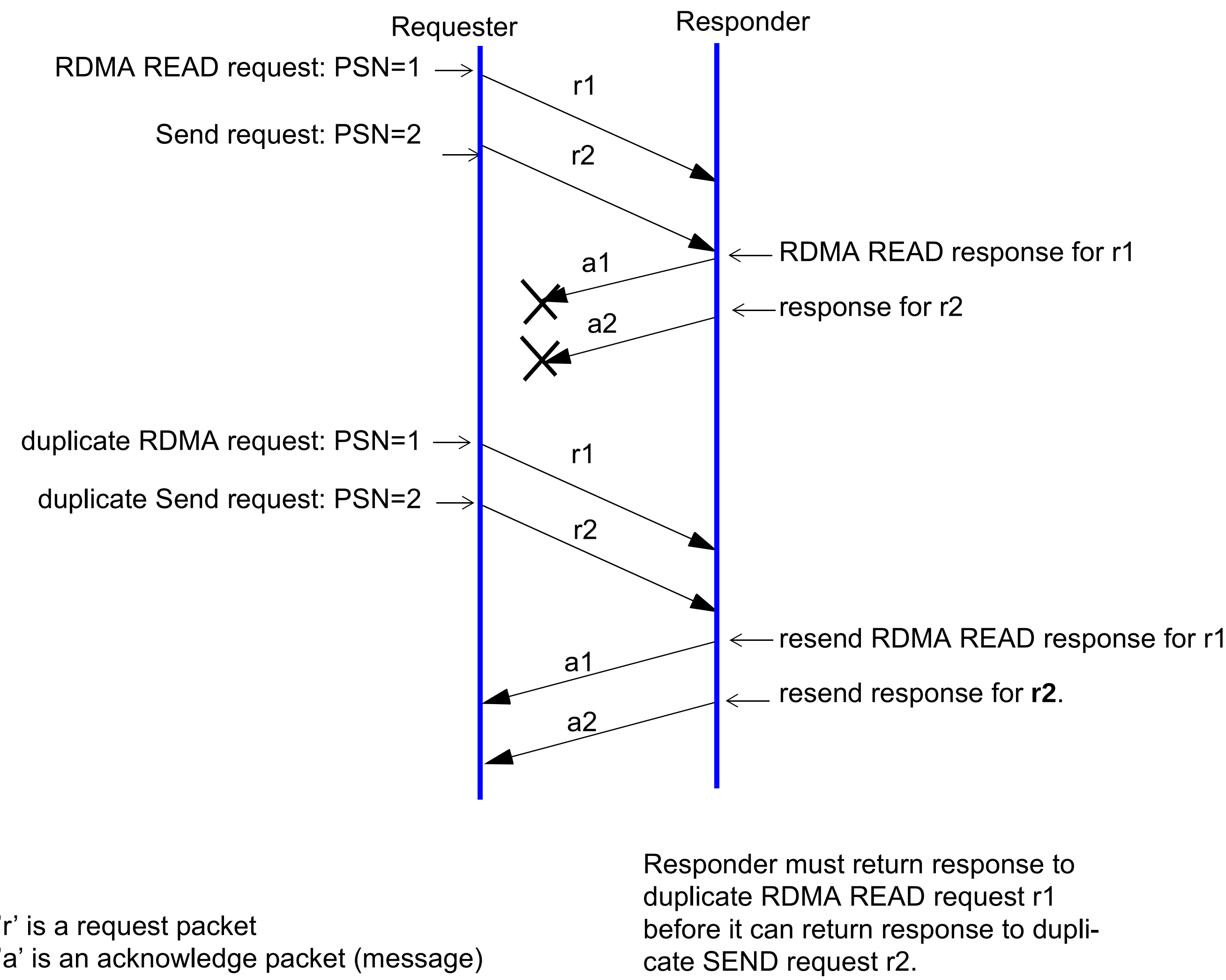


Figure 91 Maintaining the Order of Responses to Duplicate Requests

The response to be generated is a function of the duplicate request message as follows:

- SEND, RESYNC or RDMA WRITE Request

C9-105: For an HCA which receives a duplicate inbound SEND or RDMA Write request, or for a TCA which receives a duplicate inbound SEND request, the responder shall not re-execute the request but only generates a response packet for the duplicate packet, pending responses for any outstanding duplicate RDMA READ requests or ATOMIC Operation requests.

o9-64: If a TCA responder implements RDMA functionality, it shall not re-execute the RDMA WRITE request but only generate a response packet for the duplicate packet, pending responses for any outstanding duplicate RDMA READ requests or ATOMIC Operation requests.

The PSN of the acknowledge message must be the PSN of the most recently completed new request. One way to think of this process is as a logical extension of the ability to coalesce acknowledges. Indeed, the requester, on receiving a response to a duplicate request, treats it exactly as it would any other coalesced acknowledge; any outstanding duplicate RDMA READ or ATOMIC Operation requests are considered to be NAK'ed. In this case, by returning the PSN of the most recently completed request, the responder is informing the requester that it believes it has already seen and executed all requests between the duplicate request and the most recently completed request. This is illustrated in Figure 92.

C9-106: For duplicate SEND, RESYNC or RDMA WRITE requests, if the responder detects an error while in the process of returning the response, it shall silently drop the duplicate request. This is done in order to avoid confusion with any possible outstanding NAKs to new requests.

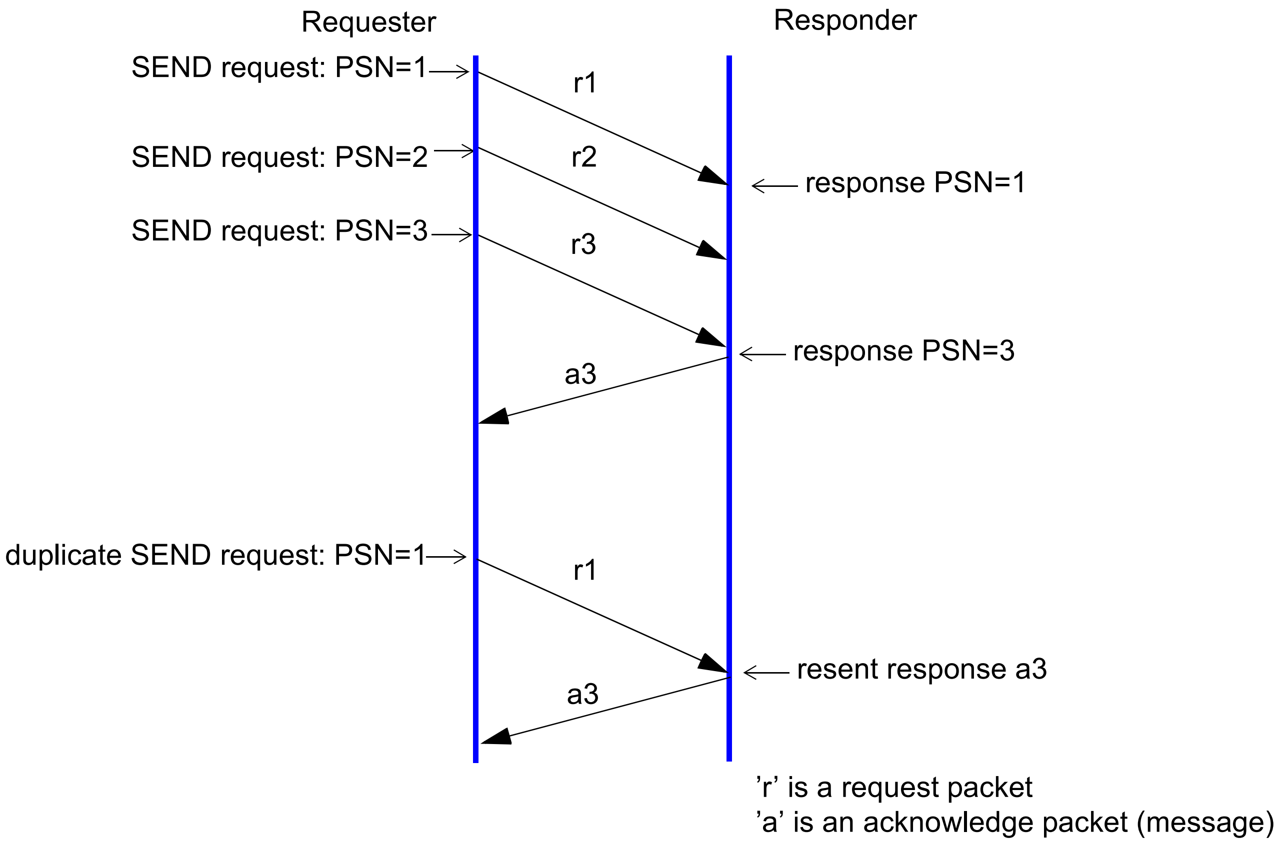


Figure 92 Acknowledging a Duplicate SEND Request

- RDMA READ Request

C9-107: An HCA responder must re-create the requested read response data. The resulting read data is returned to the requester in an RDMA READ response. The PSN of the first RDMA READ response packet shall be the same as the PSN of the duplicate request, with the PSNs for the subsequent response packets incrementing according to the normal rules for generating PSNs for RDMA READs.

C9-108: If an HCA responder detects an error while re-executing a duplicate RDMA READ request before returning the first response packet, the responder shall silently drop the duplicate request.

C9-109: If an HCA responder detects an error while re-executing a duplicate RDMA READ Request after returning one or more response packets, the RDMA READ response operation shall be aborted, i.e. no more response packets shall be returned.

When an RDMA READ Request is generated, a certain number of sequential PSN numbers are allocated based on the number of packets expected in the RDMA READ Response. These PSNs are used by the responder when generating the RDMA READ Response packet(s) Also, the original request contains a DMA Length defined in the RETH which represents the extent of the data being requested.

As described in Section 9.4.4 RDMA READ Operation on page 222, to be considered a duplicate RDMA READ Request, the PSN of the duplicate request must be within the responder's current duplicate PSN region. Furthermore, to be considered a valid duplicate RDMA READ Request, the PSN of the duplicate request must fall within the range of PSNs allocated to the original RDMA READ Response, and the amount of data requested in the duplicate request must be entirely contained within the extent of data requested in the original RDMA READ Request. In other words, the data requested in the duplicate RDMA READ Request must be a proper subset of the data requested in the original RDMA READ Request.

If the starting PSN and length of a duplicate RDMA READ Request does not fall within the range of PSNs allocated to the original RDMA READ Response, the request is invalid and the responder may silently drop the duplicate RDMA READ Request.

C9-110: A responder shall respond to all duplicate requests in PSN order; i.e. the request with the (logically) earliest PSN shall be executed first. If, while responding to a new or duplicate request, a duplicate request is received with a logically earlier PSN, the responder shall cease responding to the original request and shall begin responding to the duplicate request with the logically earlier PSN.

If a responder is interrupted by a duplicate request, it is not required to resume the interrupted request. It is the requester's responsibility to resend any unacknowledged requests.

o9-65: If a TCA implements RDMA functionality, RDMA READ Responses shall conform to the previous 4 compliance statements for HCAs.

> Following the duplicate RDMA READ response, the responder may acknowledge any subsequent duplicate Send or RDMA WRITE requests with the PSN of the most recently completed request. This is illustrated in Figure 93

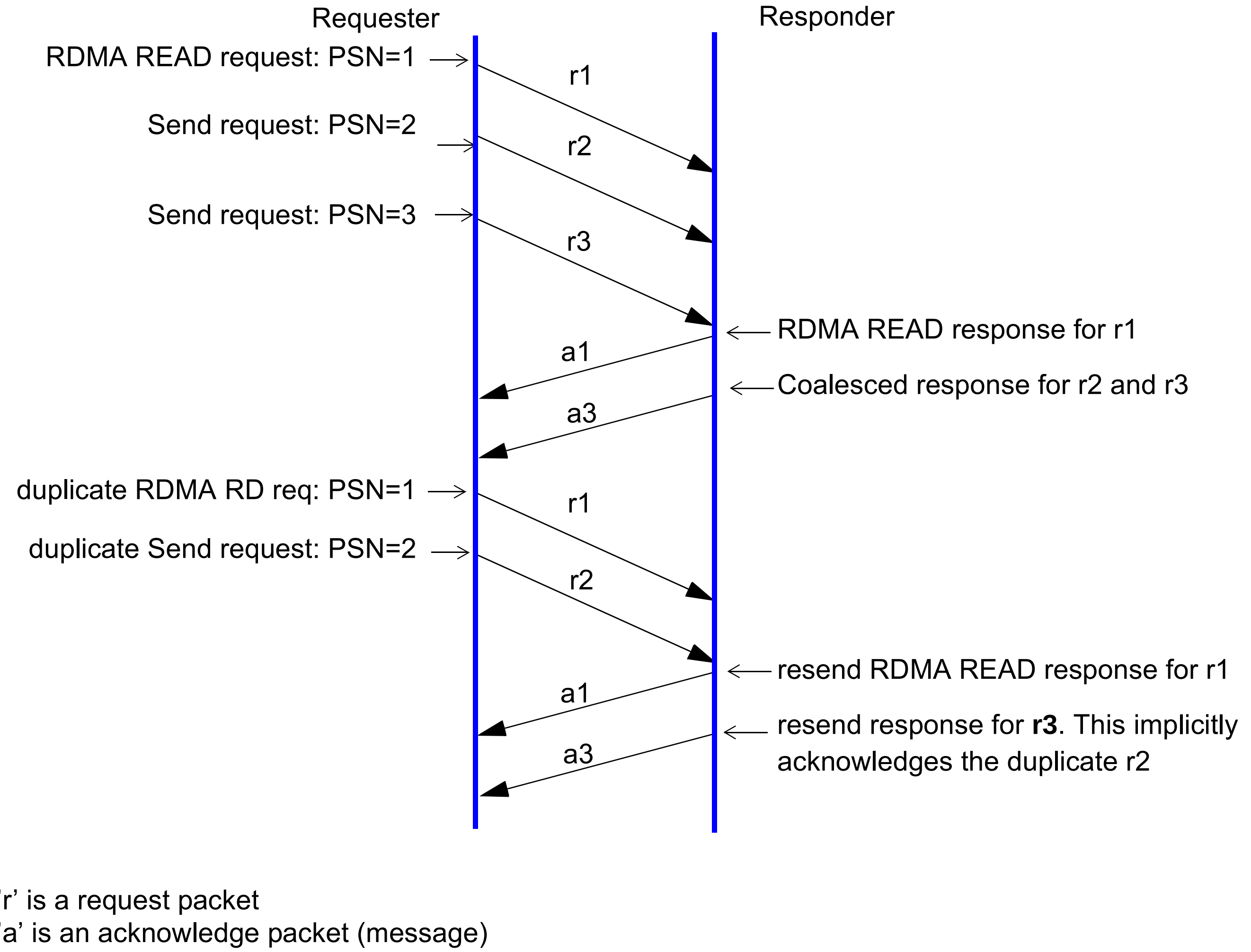


Figure 93 Acknowledging a Duplicate RDMA READ Request

- ATOMIC Operation Request

  A given receive queue may have resources to support only a limited number of ATOMIC Operations. When a duplicate ATOMIC Operation request is received, the PSN of the duplicate request is compared to the PSNs of the recently executed ATOMIC Operations.

o9-66: If the PSN of the duplicate ATOMIC Operation request matches exactly the PSN of one of the recently executed ATOMIC Operations, the saved results of that operation shall be returned to the requester. The responder shall not re-execute the request.

o9-67: If the PSN of the duplicate ATOMIC Operation request does not match the PSN of one of the recently executed ATOMIC Operations, the request is invalid and the duplicate request packet shall be silently dropped. This should never happen as long as the requester is observing the limits on the number of outstanding ATOMIC Operation requests.

o9-68: If a local error prevents the responder from reproducing the original ATOMIC Operation request data, the responder must silently drop the duplicate request.

In all cases, the PSN returned in the acknowledge message is the PSN of the duplicate request.

9.7.5.1.5 GENERATING NAKS

There are several circumstances that cause a responder to generate a NAK.

C9-111: In all cases except for RDMA READ requests, the PSN of the NAK packet shall contain the responder's expected PSN.

C9-112: In the case of an RDMA READ response packet, the PSN given in the NAK response packet shall point to the RDMA READ response packet which is being NAK'ed.

C9-113: When generating an RNR NAK, the PSN of the response packet shall point to the PSN of the packet being RNR NAK'ed.

C9-113.a1: When generating a NAK, the packet containing the NAK code shall have an opcode of Acknowledge.

This means that, even for an RDMA Read response, if the responder is returning a NAK code, it does so by substituting a packet with an opcode of Acknowledge instead of the normal opcode of RDMA Read Response (first/middle/last/only).

Once the responder has returned a NAK-sequence error or an RNR NAK, it waits for the requester to send a packet with the responder's expected PSN.

The rules that the responder must follow are as follows:

C9-114: Once a NAK has been returned for a PSN sequence error, the responder shall ignore all other new requests, except duplicate requests, until it receives a valid request with a PSN that matches its expected PSN. It shall not return any other NAK packets, except in response to a valid request with a PSN that matches its expected PSN.

C9-115: The responder must continue to respond to duplicate requests as specified above. However, the responder shall not return a NAK in response to an error condition occurring while processing a duplicate request.

9.7.5.1.6 ACKNOWLEDGE MESSAGE SCHEDULING

The scheduling of responses, per se, is not specified; however the requester may use the AckReq bit in the BTH to require the responder to schedule a response.

C9-116: When the responder receives a valid request packet with the AckReq bit set, it shall schedule a response packet for that request. The PSN of the response must be equal to or logically greater than (modulo 2^{24}) the PSN of the request.

After receiving a request message with the AckReq bit set, the responder still accepts request messages, (including additional ones marked with the AckReq bit set), while it is preparing to transmit the response packet. These additional request messages may result in a coalesced ACK when the responder is able to send the response. In this case, the single response message may satisfy several outstanding request messages.

When one or more requests arrive without the AckReq bit set, the responder may choose to deliberately coalesce its responses; it may even wait an unbounded time for additional requests, until one arrives that requires the scheduling of a response.

There are several places where the AckReq bit can be very useful to the requester. For example, if the requester is sending the last packet of the last request WQE posted to the send queue, it is advisable for the requester to set the AckReq bit or use some other mechanism to force the responder to return a response. If the requester does not do so, there is a possibility that the responder will choose to coalesce responses indefinitely. Some other mechanisms that the requester can use to ensure that the responder returns a response are:

- Always set the AckReq bit on the last (or only) packet of every message
- Follow a given request with a NOP with the AckReq bit set
- Retry the request for which a response was desired with the AckReq bit set.

For SEND or RDMA WRITE requests, an ACK may be scheduled before data is actually written into the responder’s memory. The ACK simply indicates that the data has successfully reached the fault domain of the responding node. That is, the data has been received by the channel adapter and the channel adapter will write that data to the memory system of the responding node, or the responding application will at least be informed of the failure.

The absence of the AckReq bit does not prohibit the responder from generating a response packet. As always, RDMA READ and ATOMIC Operation requests require explicit responses, thus the AckReq bit has no effect on requests.

9.7.5.1.7 RESPONSE FORMATS

Responses may take one of three forms:

1) An acknowledge packet for a normal SEND, RESYNC or RDMA WRITE operations,
2) RDMA READ responses, and
3) Acknowledge messages for ATOMIC Operations - see 9.4.5 ATOMIC Operations on page 227.

The key distinctions between the three forms is that the normal acknowledge packet (used for SENDs, RESYNCs and RDMA WRITEs) does not carry a payload field, while the responses for both the RDMA READ and ATOMIC Operations do. This observation impacts both the format of the response and the rules for coalescing acknowledges.

An acknowledge packet contains the following information:

- A syndrome used to notify the requester of the success or failure of a given request message,
- A PSN value used by the requester to correlate the acknowledge message with its listing of outstanding requests,
- A Message Sequence Number used by the responder to notify the requester that request messages have been completed,
- Optional End-to-End flow control credits,
- Payload data in the case of a RDMA READ response or ATOMIC Operation response.

Each of the three forms is discussed in the following sections.

9.7.5.1.8 RESPONSE FORMAT FOR SEND, RESYNC OR RDMA WRITE REQUESTS

This format is used to acknowledge SEND, RESYNC or RDMA WRITE request packets. A normal acknowledge message comprises a single packet, and is shown in Figure 94.

LRH	GRH	BTH	RDETH	AETH	ICRC	VCRC

note 1: GRH may or may not appear, depending on the LRH Next Header field
note 2: RDETH appears only for reliable datagram operations
note 3: DETH, RETH, EOP, PYLD and IMM fields are prohibited

Figure 94 Response Format for SENDs, RDMA WRITEs

9.7.5.1.9 RDMA READ RESPONSES

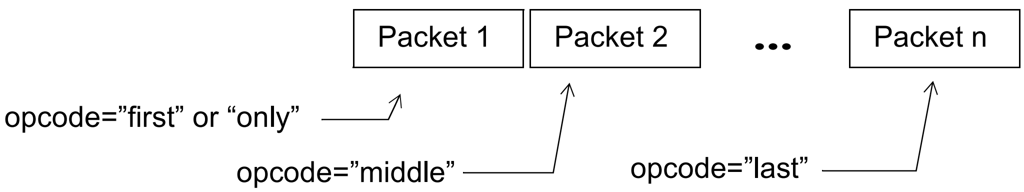
This response format, called a RDMA READ response, is used to acknowledge RDMA READ requests. A RDMA READ response message consists of one or more packets.

C9-117: For an HCA, the PSNs of the RDMA READ response packets must be sequential and monotonically increasing. If the response message consists of more than one packet, the first and last packets of the response message must contain an Acknowledge Extended Transport Header (AETH).

o9-69: If a TCA implements RDMA functionality, the PSNs of the RDMA READ response packets must be sequential and monotonically increasing. If the response message consists of more than one packet, the first and last packets of the response message must contain an Acknowledge Extended Transport Header (AETH).

C9-118: For an HCA, if the response message contains only a single packet (an “only” packet), then that packet must contain an AETH. This is illustrated in Figure 95.

o9-70: If a TCA implements RDMA functionality, and the response message contains only a single packet (an “only” packet), then that packet must contain an AETH as shown in Figure 95.



Each RDMA READ Response message comprises one or more pack-

LRH	GRH	BTH	RDETH	PYLD	ICRC	VCRC

Format for all middle packets. PYLD must be (PMTU) bytes long.

LRH	GRH	BTH	RDETH	AETH	PYLD	ICRC	VCRC

Format for first, last or only RDMA READ Response Packet.

If a first packet, PYLD shall be (PMTU) bytes long.
If an only packet, PYLD shall be zero to (PMTU) bytes long.
If a last packet, PYLD shall be one to (PMTU) bytes long.

note 1: GRH may or may not appear, depending on the LRH:Next Header field
note 2: RDETH appears only for reliable datagram operations
note 3: DETH, RETH, and IMM headers are prohibited

Figure 95 Acknowledge Message Format for RDMA READ Requests

A RDMA READ Response message, besides acknowledging the RDMA READ request itself, also implicitly acknowledges requests preceding the RDMA READ request. The rules governing coalesced ACKs are given in Section 9.7.5.1.2 Coalesced Acknowledge Messages on page 275.

The arrival of either a first packet or an only packet triggers the implicit acknowledges of any outstanding request messages as specified in section 9.7.5.1.2 Coalesced Acknowledge Messages on page 275. This is done in order to reduce the latency to complete any outstanding request messages.

The arrival of a last packet or an only packet triggers the explicit acknowledge of the RDMA READ request itself.

9.7.5.2 AETH FORMAT

Acknowledge syndromes are carried in the AETH of the acknowledge message. The table below illustrates the syndrome field of the AETH.

C9-119: When generating an AETH, a HCA responder implementing RC service shall encode the AETH Syndrome Field as shown in Table 43 AETH Syndrome Field on page 291.

o9-71: If a TCA responder implements RC service, or if a CA responder implements RD service, the responder shall encode the AETH Syndrome Field as shown in Table 43 AETH Syndrome Field on page 291.

Table 43 AETH Syndrome Field

bit 7	bits 6:5	bits 4:0	Definition
0	0 0	C CCCC	ACK (C CCCC = credit count)
0	0 1	T TTTT	RNR NAK (T TTTT = timer value)
0	1 0	X XXXX	reserved
0	1 1	N NNNN	NAK (N NNNN = NAK code)

C9-119.a1: For an HCA, the msb of the AETH Syndrome Field is reserved and shall be set to zero.

o9-71.a1: For a TCA which provides RC or RD service, the msb of the AETH Syndrome Field is reserved and shall be set to zero.

o9-72: If a CA implements Reliable Datagram service, the C CCCC bits are set to zero, since end to end credits are not defined for RD service.

The interpretation of bits [4:0] depends on the code contained in bits [6:5]. Bits [4:0] may contain a positive acknowledgment with or without end-to-end flow control credits (depending on whether the service is RC or RD), an RNR NAK timer value, a positive acknowledgment without end-to-end credits, or a NAK code.

C CCCC = encoded end-to-end flow control credits

T TTTT = RNR NAK Timer Field - see Table 45 Encoding for RNR NAK Timer Field on page 297

N NNNN = NAK Code - see Table 44 NAK Codes on page 292

Code 011 N NNNN (NAK) allows MSNs to be carried with NAK packets.

Acknowledge syndromes are carried in the AETH of the acknowledge message. The table below illustrates the syndrome field of the AETH.

9.7.5.2.1 End-to-End Flow Control Credit Field

If bits [7:5] of the AETH Syndrome field are zero, then bits [4:0] of the AETH Syndrome field carries encoded end-to-end flow control credits from the responder to the requester. This field is only valid for reliable connections.The encoding 5b11111 means that the credit field is not valid. This encoding is also used for cases where the receive queue does not support End-to-End credits. See Section 9.7.7.2 End-to-End (Message Level) Flow Control on page 314 for further details.

9.7.5.2.2 NAK Codes

If bits [6:5] of the AETH Syndrome field are b11, then bits [4:0] carry a NAK code. The code guides the requester in selecting a recovery strategy. The following sections describe all the possible NAK Codes. Even though an RNR NAK has its own AETH syndrome (AETH[6:5] = b01), RNR NAK is also described in this section.

The list of valid NAK codes is provided in Table 44.

Table 44 NAK Codes

NAK Code (AETH bits 4:0)	Definition
0 0000	PSN Sequence Error
0 0001	Invalid Request
0 0010	Remote Access Error
0 0011	Remote Operational Error
0 0100	Invalid RD Request
0 0101 - 1 1111	reserved

C9-120: If a requester receives an acknowledge message containing a reserved code, it shall consider the acknowledge packet to be malformed and shall silently drop it. This may eventually cause the requester to time out while waiting for the missing acknowledge packet, at that time it will either re-transmit the original request message, or stop operations on that send queue.

9.7.5.2.3 PSN SEQUENCE ERROR

A sequence error occurs when a responder detects a packet that is out of PSN sequence, i.e. a PSN value that is neither equal to the expected PSN nor within the valid duplicate packet range.

C9-121: The responder, when it constructs NAK packet in response to a sequence error, must insert its expected PSN value in the PSN field of the BTH. This lets the requester back up its send queue to at least the point of the failure and begin re-sending request packets from that point forward.

A PSN sequence error may be retried by the requester a number of times. Once the retry count has expired, the requester's transport notifies its client that it did not succeed in transferring the message. The requester's required behavior once its retry count has expired is given in 9.9.2 Requester Side Error Behavior on page 364. The following discussion specifies the behavior before the retry count has expired.

When the responder detects a sequence error there is no impact on the receive queue nor are any WQEs consumed. Instead, the receive queue simply returns the NAK packet to the requester and resumes waiting for an inbound request packet with the correct PSN value.

C9-122: Once a NAK packet for a sequence error has been returned to the requester, the responder shall discard all subsequent requests that do not contain the responder's expected PSN, except for valid duplicate requests.

C9-123: If the responder receives a request packet with a PSN that is logically less than its expected PSN (i.e. a valid duplicate request packet), it shall respond to that request according to the rules for duplicate packet processing.

9.7.5.2.4 REMOTE ACCESS ERROR

A R_Key violation is caused by any or all of the following conditions for either a RDMA READ, RDMA WRITE, or ATOMIC Operation:

- The R_Key field of the RETH is invalid.
- The virtual address and length or type of access specified is outside the locally defined limits associated with the R_Key.

C9-124: For an HCA responder, when reporting an RDMA remote access error, the BTH field of the acknowledge message must contain the PSN of the request packet that caused the remote access error.

o9-73: If a TCA responder implements RDMA functionality, or if a CA responder supports ATOMIC operations, then when reporting a remote access error, the BTH field of the acknowledge message must contain the PSN of the request packet that caused the remote access error.

The responder's behavior on detecting an access error, beside generating a NAK-Remote Access Error packet, is specified in section 9.9.3 Responder Side Behavior on page 375.

The requester's behavior on receiving a NAK-Remote Access Error is specified in section 9.9.2 Requester Side Error Behavior on page 364.

9.7.5.2.5 INVALID REQUEST

The requester has requested an operation that is outside the established usage of the transport service - generally, this is an OpCode that is not supported by the responder or a request whose length exceeds the available receive buffer space. For example, an RDMA request transmitted to a responder that does not support RDMAs would cause an Invalid Request Error. An out-of-sequence OpCode may also cause a NAK-Invalid Request depending on the particular service.

C9-125: When reporting an invalid request, the BTH field of the acknowledge packet must contain the responder's expected PSN value, i.e., the PSN of the request packet that contained the invalid request.

The responder's behavior upon detecting an invalid request, besides generating a NAK-Invalid Request, is given in section 9.9.3 Responder Side Behavior on page 375.

The requester's behavior on receiving a NAK-Invalid Request is given in section 9.9.2 Requester Side Error Behavior on page 364.

9.7.5.2.6 REMOTE OPERATIONAL ERROR

A remote operational error occurs when the responder encounters a situation that prevents its receive queue from completing the current request. The list of error conditions detectable by the responder, and reportable as a remote operational error, is not specified since it is implementation specific. Remote operational errors cannot be caused by anything the requester may have done. Rather, they reflect a fault in the responder.

C9-126: When reporting a remote operational error, the BTH field of the acknowledge message must contain the PSN of the request being executed at the time the responder detected the operational error.

The responder's behavior upon detecting an operational error, besides returning NAK-Remote Operational Error, is given in section 9.9.3 Responder Side Behavior on page 375.

The requester's behavior when it receives a NAK-Remote Operational Error is specified in section 9.9.2 Requester Side Error Behavior on page 364.

9.7.5.2.7 INVALID RD REQUEST

This NAK code is generated when the responder detects a Q_Key or RDD violation while operating in RD service, or if the destination QP is not configured for RD service, or if the destination QP is not in a state where it can accept an inbound packet.

o9-74: If the responder's EE Context detects an invalid P_Key, the request packet shall be silently dropped by the EE Context.

If no P_Key violation is detected, the EE Context forwards the packet to the receive queue specified in the BTH.

C9-127: If the QP as specified in the BTH is not configured for RD service, then a NAK-Invalid RD Request shall be returned.

The receive queue checks the Q_Key of the inbound request packet and also checks that its current RDD value matches that of the EE Context.

If the responder's receive queue detects an invalid Q_Key, or if the receive queue's RDD value does not match that of the EE Context, the responder shall return a NAK-Invalid RD Request to the requester.

The responder's behavior upon detecting either a Q_Key or RDD violation, beside generating a NAK-Invalid RD Request, is specified in section 9.9.3 Responder Side Behavior on page 375.

The requester's behavior in response to a NAK-Invalid RD Request is specified in section 9.9.2 Requester Side Error Behavior on page 364.

9.7.5.2.8 RNR NAK

Under certain circumstances, a receive queue may be temporarily unable to accept an inbound request message. For example, there may not currently be a valid receive WQE posted to the receive queue. When this occurs, the responder may return a response indicating Receiver Not Ready (RNR NAK). Note: the responder may return an RNR NAK for any type of request (e.g. SEND, RDMA READ request, RDMA WRITE request, etc.). On receiving a RNR NAK, the requester may, after waiting for at least the interval specified in the RNR NAK, retry the same request. "The same request" means the precise same request message beginning with the same PSN as reported by the responder in its RNR NAK packet.

C9-128: For an HCA requester using Reliable Connection service, after receiving an RNR NAK, the requester shall not substitute a different request message by reusing the same PSN.

o9-75: If a TCA requester implements Reliable Connection service, after receiving an RNR NAK, the requester shall not substitute a different request message by reusing the same PSN.

For Reliable Datagram service, the requester may either exactly repeat the request, move the EEC to the Error state, abandon the request or suspend the request as described in Section 9.7.8 Reliable Datagram on page 323.

C9-129: An HCA responder using Reliable Connection service, when generating an RNR NAK, shall indicate the appropriate interval in the timer field of the AETH. The value loaded in the timer field of the AETH shall be as shown in Table 45 Encoding for RNR NAK Timer Field on page 297.

o9-76: If a CA responder implements Reliable Datagram service, or if a TCA implements Reliable Connection service, it shall follow this rule: when generating an RNR NAK, the responder shall indicate the appropriate interval in the timer field of the AETH. The value loaded in the timer field of the AETH shall be as shown in Table 45 Encoding for RNR NAK Timer Field on page 297.

C9-130: An HCA requester providing Reliable Connection service, after receiving a RNR NAK, must wait for at least the interval specified in the timer field of the AETH before retrying the request. If the requester fails to wait for the appropriate timeout interval before re-trying the request, the responder may silently drop the packet.

o9-76.a1: An HCA requester providing Reliable Datagram service, after receiving a RNR NAK, must wait for at least the interval specified in the timer field of the AETH before retrying the request. If the requester fails to wait for the appropriate timeout interval before re-trying the request, the responder may silently drop the packet.

o9-76.a2: A TCA requester providing Reliable Connection service, or a TCA requester providing Reliable Datagram service, after receiving a RNR NAK, must wait for at least the interval specified in the timer field of the AETH before retrying the request. If the requester fails to wait for the appropriate timeout interval before re-trying the request, the responder may silently drop the packet.

C9-131: This compliance statement is obsolete and has been removed.

C9-132: An HCA requester using Reliable Connection service shall maintain a 3 bit retry counter which is loaded during connection establishment with information provided by the responder. This counter is used to limit the number of times a requester can retry an operation which was RNR NAK'ed. When a RNR NAK response is received, if the RNR NAK retry counter is not equal to 7 (indicates infinite retry), the requester shall decrement the RNR NAK retry counter. Thereafter, when the retry timer expires, if the retry counter is non-zero, the requester may re-issue the request.

o9-77: If a CA requester implements Reliable Datagram service, or if a TCA requester implements Reliable Connection Service, it shall maintain a 3 bit retry counter which is loaded during connection establishment with information provided by the responder. This counter is used to limit the number of times a requester can retry an operation which was RNR NAK'ed. When a RNR NAK response is received, if the RNR NAK retry counter is not equal to 7 (indicates infinite retry), the requester shall decrement the RNR NAK retry counter. Thereafter, when the retry timer expires, if the retry counter is non-zero, the requester may re-issue the request.

A locally detected error is recorded by the requester if the retry counter has decremented to zero at the time that the RNR NAK retry timer expires. See Section 9.9.2.1 Requester Side Error Detection - Locally Detected Errors on page 364 for further details.

The timer field is encoded as shown in the Table below.

Table 45 Encoding for RNR NAK Timer Field

RNR Time	Delay in milliseconds	RNR Time	Delay in milliseconds
00000	655.36	10000	2.56
00001	0.01	10001	3.84
00010	0.02	10010	5.12
00011	0.03	10011	7.68
00100	0.04	10100	10.24
00101	0.06	10101	15.36

Table 45 Encoding for RNR NAK Timer Field

RNR Time	Delay in milliseconds	RNR Time	Delay in milliseconds
00110	0.08	10110	20.48
00111	0.12	10111	30.72
01000	0.16	11000	40.96
01001	0.24	11001	61.44
01010	0.32	11010	81.92
01011	0.48	11011	122.88
01100	0.64	11100	163.84
01101	0.96	11101	245.76
01110	1.28	11110	327.68
01111	1.92	11111	491.52

The use of RNR NAK for temporary problems that do not affect the whole message (such as a memory page not present) is not prohibited. In particular, for Reliable Datagram service, an RNR NAK returned in the middle of a SEND request message by a responder may result in the current message being abandoned by the requester and a new message being sent from another queue pair. This may result in unexpected incomplete messages at the responder. These incomplete messages are detected by the responder while executing a RESYNC request, thus allowing the responder to complete the partially completed WQE in error and begin receiving the new request.

A responder should use this feature as a mechanism to delay the incoming request when a local resource is unavailable only rarely. The RNR NAK mechanism consumes bandwidth in that an incoming packet will be aborted and will have to be re-sent.

9.7.6 REQUESTER: RECEIVING RESPONSES

9.7.6.1 VALIDATING INBOUND RESPONSE PACKETS

On receipt of an inbound acknowledge packet, a requester validates the packet as follows:

C9-133: To verify the integrity of the packet, the requester shall validate the packet as specified in Section 9.6 Packet Transport Header Validation on page 236. Invalid packets shall be silently dropped by the requester.

C9-134: For an HCA requester using Reliable Connection service, the PSN shall be examined to detect out of order packets. Since acknowledges may be coalesced as described in section 9.7.5.1.2 Coalesced Acknowledge Messages on page 275, the PSN is used to detect coalesced responses.

o9-78: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the PSN of each acknowledge packet shall be examined to detect out of order packets. Since acknowledges may be coalesced as described in section 9.7.5.1.2 Coalesced Acknowledge Messages on page 275, the PSN is used to detect coalesced responses.

C9-135: For an HCA requester using Reliable Connection service, the validity of the acknowledge syndrome shall be checked according to the table in Section 9.7.5.2.2 NAK Codes on page 292. A response packet containing a reserved NAK code shall be simply dropped.

o9-79: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the validity of the acknowledge syndrome shall be checked according to the table in Section 9.7.5.2.2 NAK Codes on page 292. A response packet containing a reserved NAK code shall be simply dropped.

o9-79.a1: If a CA implements Reliable Datagram service, when receiving a response packet, the requester shall check the destination QPn contained in the BTH against the expected QPn for the current EEC. If the response packet is not destined for the currently active requester side QP, it shall be dropped by the requester.

The requester must also insure that responses are appropriate for the type of operation being performed. For example, in certain congested fabric cases it is possible for an ACK to arrive when an RDMA READ or Atomic response is expected. This can happen even when the PSN contained in the response packet matches the requester's expected response PSN. The converse is also true.

C9-135.a1: The requester shall validate that the response received is consistent with the outstanding request. If it is not, the response packet shall be ignored.

If the packet is determined to be valid, it is processed by the requester. While processing the acknowledge packet, the requester may encounter local errors. The list of local errors that the requester may encounter when processing the acknowledge message is not specified since it is implementation specific, but includes any error due to a fault on the requester side. The required behaviors for this case are specified in 9.9.2 Requester Side Error Behavior on page 364.

Validating the PSN of an inbound response packet relies on identifying three critical points in the PSN sequence. These three points are:

1) Requester's maximum forward progress - the logically largest (modulo 2^{24}) PSN of any request sent by the requester. (This in-

cludes PSN space allocated for RDMA READ responses.) It marks the "righthand" edge of the Valid Region.

2) Oldest unacknowledged request - the PSN of the oldest outstanding (unacknowledged) request. This represents the maximum forward progress made by the responder, as viewed by the requester. The significance of this PSN is that it marks the end of the duplicate region, i.e. responses with PSNs logically less than this are treated either as invalid or as duplicates.

3) Oldest valid request - this point marks the "lefthand" edge of the Valid Region.

See the figure below for an illustration of these three points.

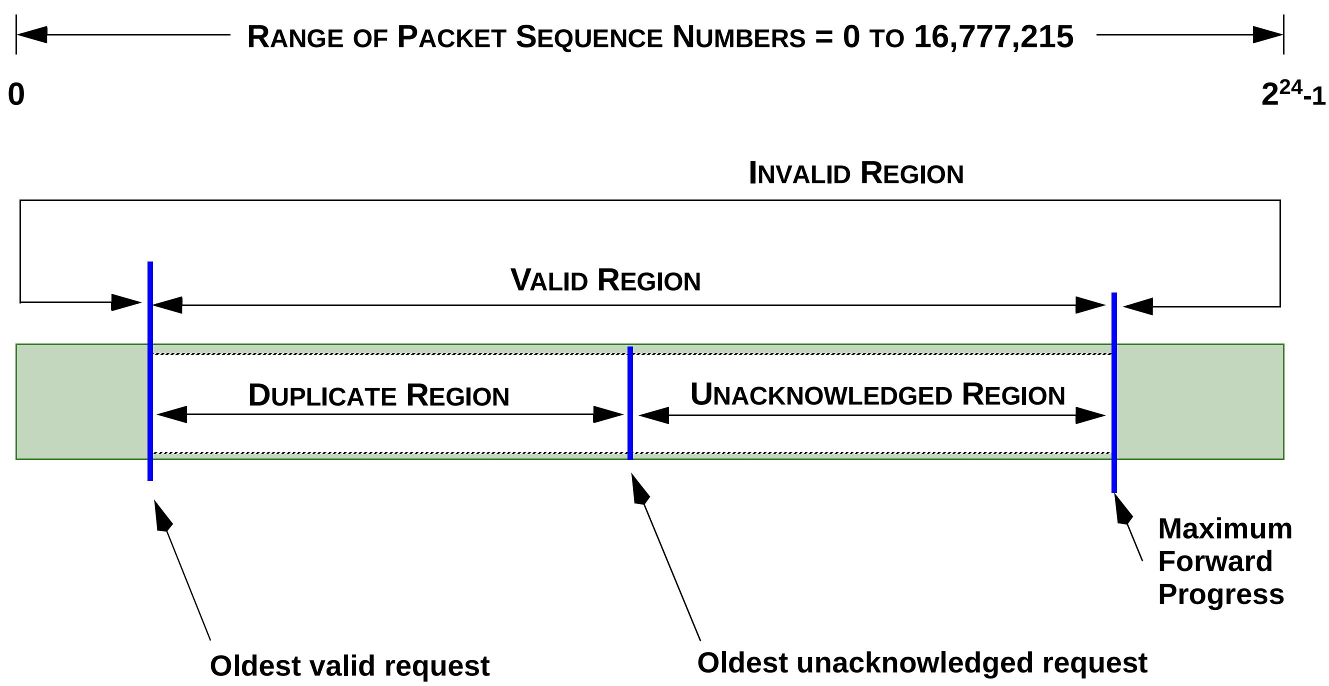


Figure 96 Response Packet PSN Regions

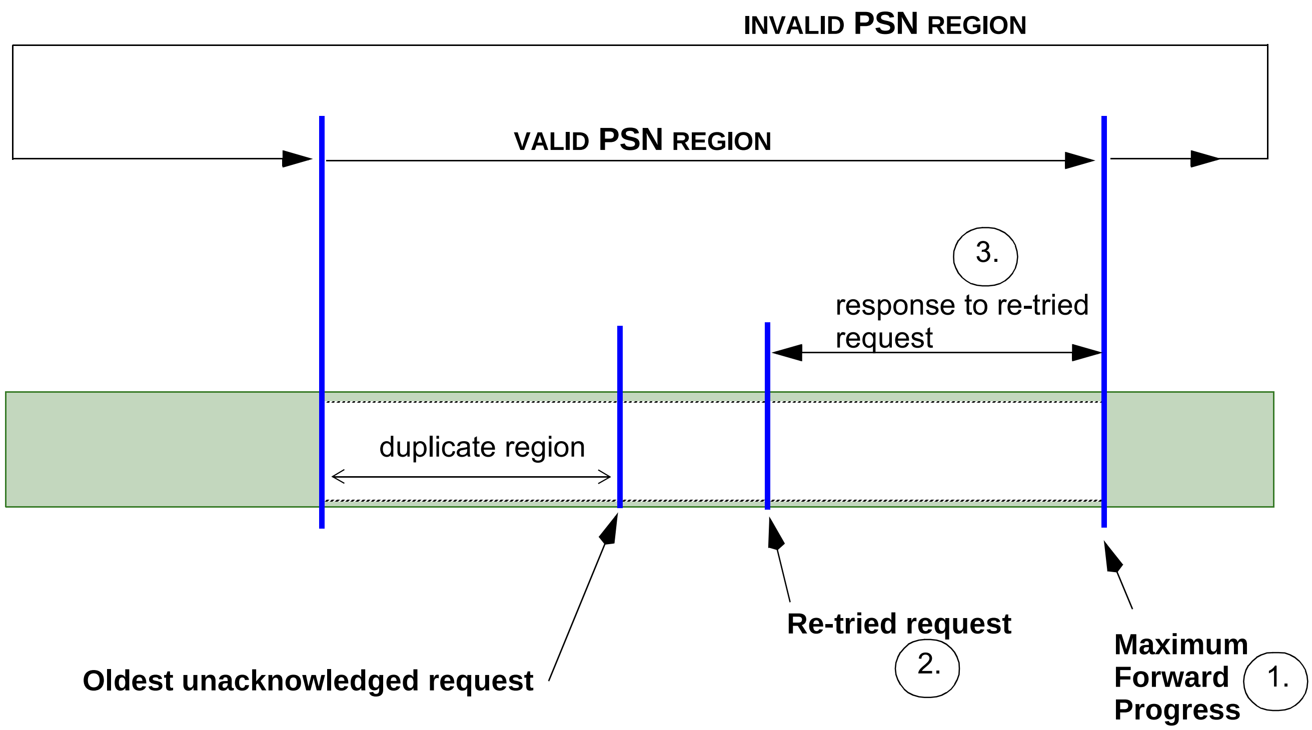
As is the case with request packets, each response packet carries a PSN. The requester, on receiving a response packet, checks the PSN to determine if the response is an expected response or a ghost acknowledge packet. Conceptually, the requester keeps track of the PSN of the oldest unacknowledged request packet and the logically largest (modulo 2^{24}) PSN sent to date including PSNs reserved for RDMA READ responses. These two PSNs define the endpoints of a range of PSNs identifie in the figure above as the Unacknowledged Region. If the PSN of a response packet falls within that range then the packet is an expected response packet. If the response does not fall within that region, then it is considered either a duplicate response and is handled according to the rules defined in Section 9.7.5.1.4 Acknowledging Duplicate Requests on page 279, or it a ghost (invalid) acknowledge packet and is dropped by the requester.

C9-136: For an HCA requester using Reliable Connection service, ghost acknowledge packets shall be dropped by the requester.

o9-80: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, ghost acknowledge packets shall be dropped by the requester.

9.7.6.1.1 PSNs for Retried Requests

Under some circumstances (described below) the requester may need to retry a request. In PSN terms, this means that the requester may re-send a packet with a PSN which is logically less (modulo 2^{24}) than the maximum PSN transmitted to date. In the figure below, this is identified as the Re-tried request.



1. Requester advances to the point of maximum forward progress.

2. Due to timeout, the request "backs up" and re-tries a request.

3. Meanwhile, the responder may have continued to make progress. Thus, the response it returns may have a PSN logically greater than the PSN of the re-tried request. (However, it cannot be greater than the requester's point of maximum forward progress.)

Figure 97 Three PSN Paradigm

During the process of retrying a request, the requester should maintain a means of marking the point of furthest PSN advance (high water mark) even though it is logically "backing up" the PSN sequence when it re-tries a request. This is necessary because the response to the re-tried request may have a PSN which is logically greater than the PSN of the re-tried request. This can happen because under some circumstances the responder might interpret the re-tried request as a duplicate request. If so, it (the responder) is obligated to return the PSN marking its furthest point of advance, which may be beyond the PSN of the re-tried request.

The need for the requester to maintain these three pointers (point of maximum advance, PSN of the oldest unacknowledged request, and the PSN of the re-tried request) is referred to as the three PSN paradigm. Although various implementations may find different ways of implementing the three PSN paradigm, nonetheless, a requester must account for the fact that a responder may return a response to a re-tried request with a PSN which is logically greater (further advanced) than the PSN of the re-tried request itself.

9.7.6.1.2 Requester Response to a NAK Message

The requester's reaction to a negative response message depends on the NAK code that is returned, and whether the queue pair is configured for reliable connected or reliable datagram service.

A NAK-Sequence error triggers an automatic retry of the request. The PSN in the response packet is the requester's indication of the request packet that the responder believes it missed, thus, the requester can retry that request. To prevent the requester from retrying the same request forever, the requester maintains a 3 bit retry counter which is used to count the number of times a particular request packet has been retried and timed out. See Section 9.9.2.1 Requester Side Error Detection - Locally Detected Errors on page 364 for a full description of the retry counter.

C9-137: An HCA requester using Reliable Connection service shall decrement its 3 bit retry counter each time the responder returns a NAK-Sequence error for a given request packet. The counter shall be re-loaded whenever the given outstanding request is cleared. If automatic path migration is not supported, and if a NAK-Sequence error is returned once more, then the requester shall declare a locally detected error.

o9-80.a1: An HCA requester which supports Reliable Datagram service shall decrement its 3 bit retry counter each time the responder returns a NAK-Sequence error for a given request packet. The counter shall be reloaded whenever the given outstanding request is cleared. If automatic path migration is not supported, and if a NAK-Sequence error is returned once more, then the requester shall declare a locally detected error.

o9-81: A TCA requester which implements Reliable Connection service or Reliable Datagram service shall decrement its 3 bit retry counter each time the responder returns a NAK-Sequence error for a given request packet. The counter shall be re-loaded whenever the given outstanding request is cleared. If automatic path migration is not supported, and if a NAK-Sequence error is returned once more, then the requester shall declare a locally detected error.

o9-82: If a CA supports automatic path migration, then the following is required. If a NAK-Sequence error is returned after the retry counter has decremented to zero, then the channel adapter shall attempt an automatic path migration. Following the automatic path migration, the requester shall reload the retry counter and begin the process over again. If the requester still does not succeed in sending the request after several retries, then the requester shall declare a locally detected error.

For other NAK packets, the response of the send queue depends on whether the queue is providing Reliable Datagram or Reliable Connected service.

Reliable Datagram Behavior: Reliable datagrams require the use of an EE Context that maintains the packet sequence numbers and thus ensures reliable delivery of requests. The rules for responding to a NAK ensure that the current PSN at the requester and the expected PSN at the responder remain in sync. Therefore, the connection between the requester's EE Context and responder's EE Context survives. This allows the connection to continue to service other Send/Receive QPs.

Depending on the cause and the operation in question, the EE context may undertake any of the following options after detecting a failed request packet:

a) It may retry the same failed packet from the same QP (note that not all NAKs can be retried, and for those that can be retried there are limits to the number of times a retry is possible), or

b) It may transition the QP and the EE Context to the Error state, completing the failed request message in error, or

c) It may place the current QP in the SQEr state and generate a RESYNC request according to the rules detailed in Section 9.7.8 Reliable Datagram on page 323. In this case, the failed WQE will usually end up being completed in error.

This last strategy is designed to allow the requester side EE Context to continue in service, thus avoiding the need to tear down the EE Context-to-EE Context connection.

If the "same failed packet" is to be retried, the requester is not required to begin its retransmission sequence beginning with the PSN indicated in the responder's NAK; instead, it may begin its retransmission with an earlier request packet. These earlier request packets are treated by the responder as normal duplicate packets causing no ill side effects.

See section 9.9 Error detection and handling on page 362 for a complete description of the errors and the EE Context's subsequent behavior.

C9-138: For an HCA requester using Reliable Connection service, the requester must receive and discard any duplicate acknowledge messages with no ill side effects.

o9-83: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, the requester must receive and discard any duplicate acknowledge messages with no ill side effects.

Reliable Connected Behavior: For reliable connections, the requester has only two possible alternatives when it receives a NAK. It may either retry the same request packet, or it may mark the current WQE as completed in error and notify its client. Note that not all NAKs can be retried.

If the requester retries the same request packet, it is not required to begin its retransmission sequence beginning with the PSN indicated in the responder's NAK; instead, it may begin its retransmission with an earlier request packet. These earlier request packets are treated by the responder as normal duplicate packets causing no ill side effects.

9.7.6.1.3 DETECTING LOST ACKNOWLEDGE MESSAGES AND TIMEOUTS

Under some error conditions the requester does not receive an acknowledge message in response to one or more of its requests. This can occur for one of three reasons:

1) The responder generated an acknowledge message that was subsequently lost in the fabric, or,
2) The responder failed for some reason preventing it from generating an acknowledge message, or
3) The original request message was lost in the fabric before it was received by the responder.

All three of these conditions are detected by the requester as a lost acknowledge message.

Often, these errors are corrected automatically due to acknowledge coalescing; the next acknowledge received by the requester serves to implicitly acknowledge all outstanding requests.

However, there are several cases where a lost acknowledge message is not automatically recovered by the coalesced acknowledge rules. For example, a NAK message lost in the fabric will not be resolved via acknowledge coalescing because the responder side rules require that the responder may have no more than one NAK message outstanding at a given time.

C9-139: For an HCA requester using Reliable Connection service, to detect missing responses, every Send queue is required to implement a Transport Timer to time outstanding requests.

o9-84: If a TCA requester implements Reliable Connection service, to detect missing responses, every Send queue is required to implement a Transport Timer to time outstanding requests.

o9-85: If a CA requester implements Reliable Datagram service, to detect missing responses, every EE Context is required to implement a Transport Timer to time outstanding requests.

Because of variabilities in the fabric, scheduling algorithms and architecture of the channel adapters and many other factors, it is not possible, nor desirable, to time outstanding requests with a high degree of precision. Nonetheless, the Transport Timer is an integral element of the ACK/NAK protocol by providing a deterministic means to detect lost requests or responses.

The requester need not separately time each request launched into the fabric, but instead simply begins the timer whenever it is expecting a response. Once started, the timer is restarted each time an acknowledge packet is received as long as there are outstanding expected responses. The timer does not detect the loss of a particular expected acknowledge packet, but rather simply detects the persistent absence of response packets.

The timer measures the lesser of:

- the time since the requester sent a packet with the AckReq bit set in the BTH,
- or the time since the last valid acknowledge packet arrived.

The operation is as follows.

The requester starts the timer running whenever the timer is not currently running AND:

1) The requester sets the AckReq bit in a Send or RDMA WRITE request or,
2) The requester generates an RDMA READ request or,
3) The requester generates an ATOMIC Operation request.

Thereafter, the requester restarts the timer each time it receives a new inbound acknowledge packet as long as there are still outstanding expected responses.

The timer is stopped whenever there are no outstanding expected responses.

An “expected response” is created by the requester by setting the AckReq bit in a request packet or by generating an RDMA READ request or an

ATOMIC Operation request. An outstanding expected response is a response to any request packet which has the AckReq bit set in the BTH, or any RDMA READ request or ATOMIC Operation request, which has not been acknowledged.

Each QP has a single Local ACK Timeout value associated with it which is used to derive the Transport Timer timeout interval $T_{tr.}$

C9-140: For an HCA requester using Reliable Connection service, the Transport Timer timeout interval, T_{tr} shall be defined to be 4.096 uS * $2^{(Local\ ACK\ Timeout)}$. Local ACK Timeout shall be a 5 bit value, with zero meaning that the timer is disabled. The minimum acceptable value of Local ACK Timeout, other than zero, shall be defined by the CA vendor. If a non-zero Local ACK Timeout value is loaded in QP context which is less than the minimum supported by the CA, then the CA may use its minimum value.

C9-141: For an HCA requester using Reliable Connection service, a QP shall provide facilities to detect a timeout condition.

The timeout interval should begin after a request has been scheduled. The timeout condition, To, should be detected in no less than the timeout interval, Tr, and no more than four times the timeout interval, 4Tr.

Thus, $T_{tr} <= T_o <= 4T_{tr.}$

Once a timeout for a given request packet is detected, the requester may retry the request.

C9-142: For an HCA requester using Reliable Connection service, to prevent the requester from retrying the request forever, the requester shall maintain a 3 bit retry counter which is used to count the number of times a particular request packet has been retried and timed out. This counter shall be decremented each time the transport timer expires for a given request packet. The counter shall be re-loaded whenever a given outstanding request is cleared.

See Section 9.9.2.1 Requester Side Error Detection - Locally Detected Errors on page 364 for a full description of the retry counter.

C9-143: For an HCA requester using Reliable Connection service, if automatic path migration is not supported, and if the transport timer expires after the retry counter has decremented to zero, then the requester shall declare a locally detected error.

o9-86: If automatic path migration is supported, and If the transport timer expires after the retry counter has decremented to zero, then the channel adapter shall attempt an automatic path migration. Following the automatic path migration, the requester shall reload the transport timer retry counter and begin the process over again. If the requester still does not succeed in sending the request after several retries, then the requester shall declare a locally detected error.

o9-87: If a TCA requester implements Reliable Connection service, then the five preceding **HCA** compliance statements (that is, timeout rules for outstanding requests) shall be applicable to that TCA.

o9-88: If a CA requester implements Reliable Datagram service, then the five preceding **HCA** compliance statements (that is, timeout rules for outstanding requests) shall be applicable to that CA. In that case, the functionality described applies to the EE Context rather than the Queue Pair.

9.7.6.1.4 DUPLICATE ACKNOWLEDGEMENTS

9.7.1 Packet Sequence Numbers (PSN) on page 248 describes how duplicate requests are generated. These requests may result in duplicate acknowledgments being returned by the responder. The responder may also send unsolicited Acks that appear to be “Ghost Acks” from the point of view of the requestor.

C9-144: For an HCA requester using Reliable Connection service, if the responder is configured to generate end-to-end flow control credits, then the requester must extract end-to-end flow control credits from a duplicate acknowledgment.

o9-89: If a TCA implements Reliable Connection service, and if the responder is configured to generate end-to-end flow control credits, then the requester must extract end-to-end flow control credits from a duplicate acknowledgment.

C9-145: For an HCA requester using Reliable Connection service, duplicate acknowledgments shall be discarded.

o9-90: If a TCA requester implements Reliable Connection service, duplicate acknowledgments shall be discarded.

See Section 9.7.7.2 End-to-End (Message Level) Flow Control on page 314 for a complete description.

9.7.7 RELIABLE CONNECTIONS

A reliable connection is a connection created between a single local QP and a single remote QP and that can guarantee that messages are delivered at most once, in order and without corruption (in the absence of unrecoverable errors) between the local and remote QPs.

Table 46 Reliable Connected Service Characteristics

Property / Level of Reliability	Support
Corrupt data detected	Yes
Data delivered exactly once (Except for an unrecoverable error - that is reported to the application)	Yes
Data order guaranteed	Yes
Data loss detected	Yes
RDMA Support	Yes - both Read and Write
State of Send/RDMA WRITE when request completed	Completion on remote end node
ATOMIC Support	Optional
Multi-packet message support	Yes
Number of messages in flight per QP	2^{23} (maximum)
Number of packets allowed in flight per QP	2^{23} (maximum)
Number of messages enqueued per QP	Implementation limited only
Maximum Message Size	2^{31}Bytes

The desired reliability characteristics are provided by application of packet sequence numbers and the ACK/NAK protocol.

C9-146: For an HCA, each QP configured for Reliable Connection service must conform to the requirements specified in section 9.7 Reliable Service on page 247, the characteristics given in Table 46 Reliable Connected Service Characteristics on page 309, and any additional requirements given in this section.

o9-91: If a TCA implements Reliable Connection service, each QP configured for Reliable Connection service must conform to the requirements specified in section 9.7 Reliable Service on page 247, the characteristics given in Table 46 Reliable Connected Service Characteristics on page 309, and any additional requirements given in this section.

9.7.7.1 Generating MSN Value

For Reliable Connected service, the Message Sequence Number is a number returned by the responder to the requester indicating the number of messages completed by the responder. The MSN is carried in the three least significant bytes of the AETH. The MSN is provided to the requester as a service to assist it in completing WQEs by informing the requester of the messages that have been completed by the responder.

C9-147: An HCA responder using Reliable Connection service shall return an MSN in the AETH of any response packet which contains an AETH.

o9-92: If a TCA responder implements Reliable Connection service, it shall return an MSN in the AETH of every response packet.

Logically, the requester associates a sequential Send Sequence Number (SSN) with each WQE posted to the send queue. The SSN bears a one-to-one relationship to the MSN returned by the responder in each response packet. Therefore, when the requester receives a response, it interprets the MSN as representing the SSN of the most recent request completed by the responder to determine which send WQE(s) can be completed.

Note that SSN as described above is a logical concept only which is given to convey the concept of how the MSN is applied; an implementation is not required to implement it as described.

Following initialization, the first WQE posted to the Send queue has an SSN of one assigned to it. The responder initializes its MSN counter to zero. Thereafter, the responder increments its 24-bit MSN value whenever it completes execution of an inbound request message. This is illustrated in Figure below.

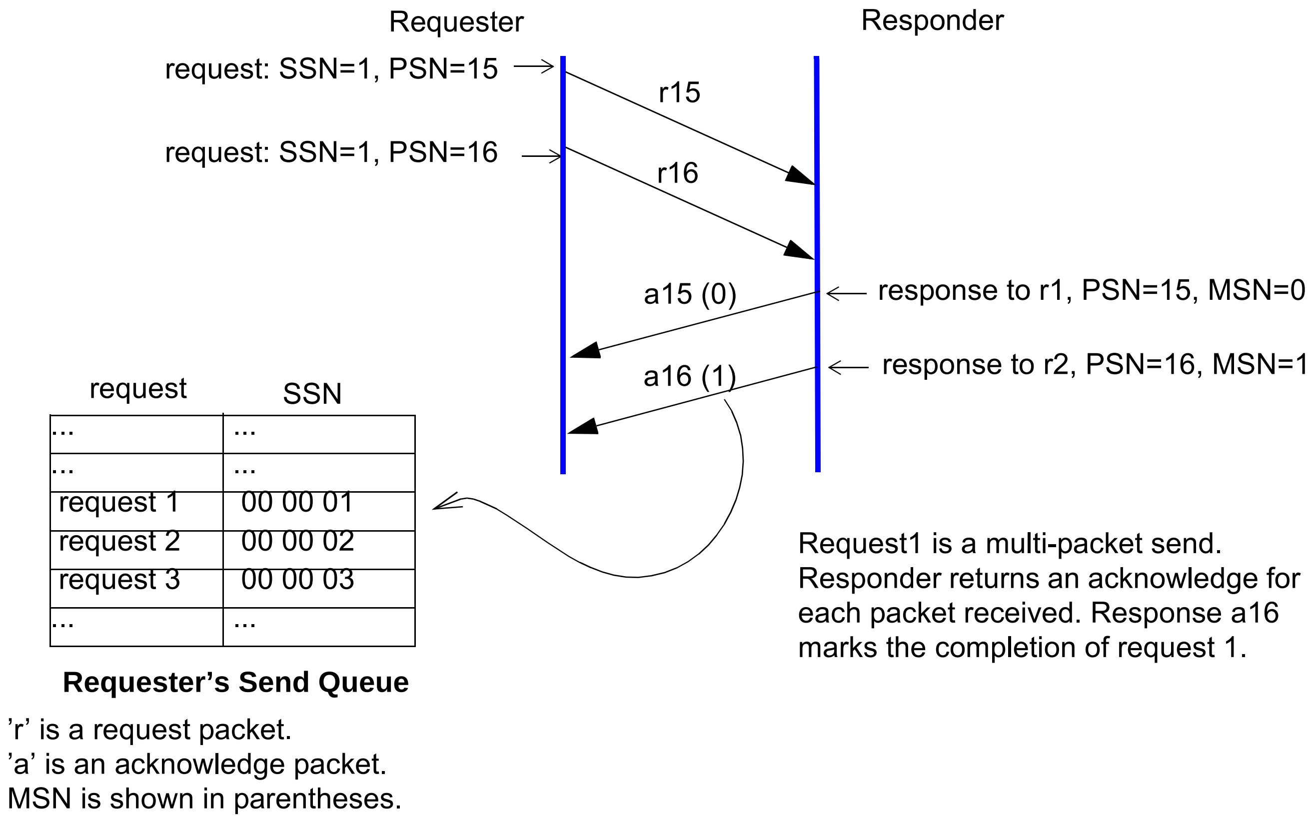


Figure 98 Responder Initializes MSN to Zero

C9-148: An HCA responder using Reliable Connection service shall initialize its MSN value to zero. The responder shall increment its MSN whenever it has successfully completed processing a new, valid request message. The MSN shall not be incremented for duplicate requests. The incremented MSN shall be returned in the last or only packet of an RDMA READ or Atomic response. For RDMA READ requests, the responder may increment its MSN after it has completed validating the request and before it has begun transmitting any of the requested data, and may return the incremented MSN in the AETH of the first response packet. The MSN shall be incremented only once for any given request message.

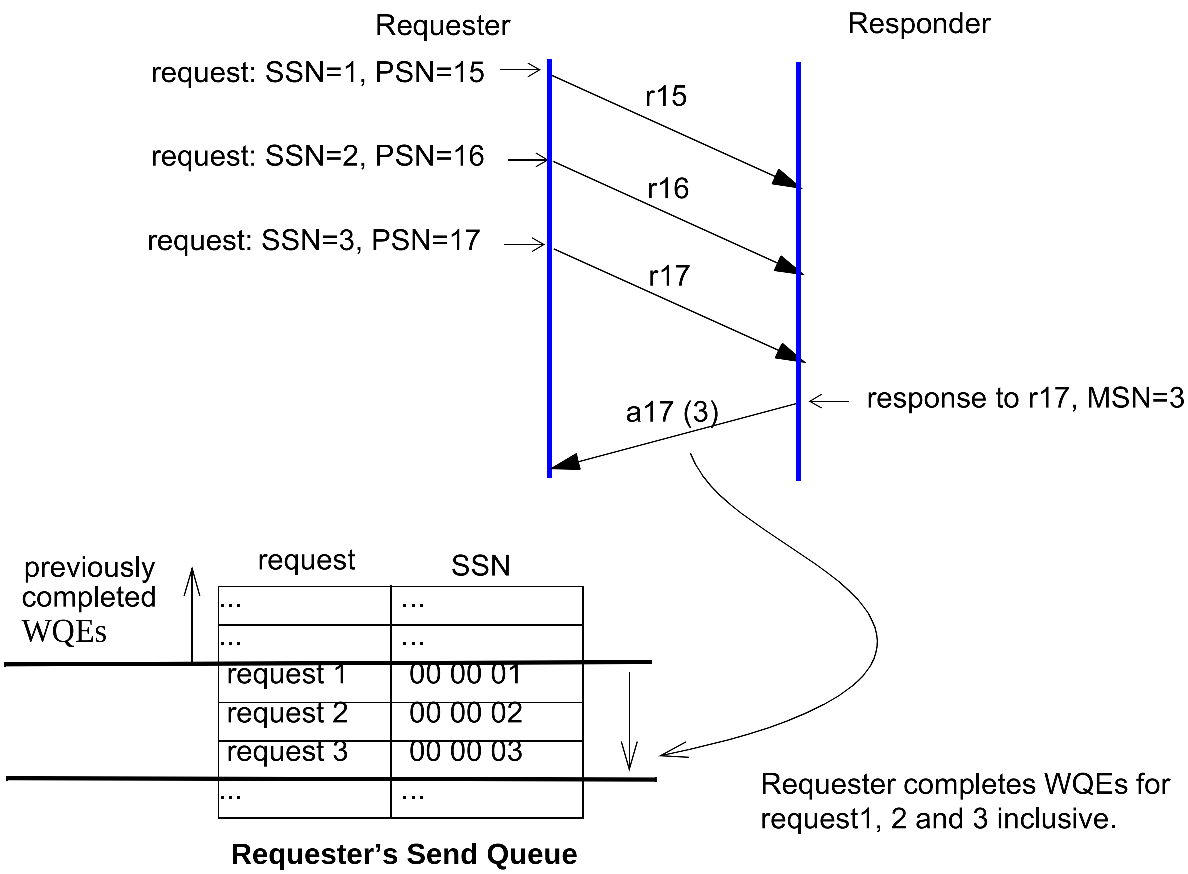
o9-93: If a TCA responder implements Reliable Connection service, it shall calculate and update MSN as described in the preceding compliance statement.

9.7.7.1.1 REQUESTER BEHAVIOR ON RECEIVING A NEW MSN

As described above, the existence of a new MSN value in a response packet may be used by the requester as a signal to complete certain

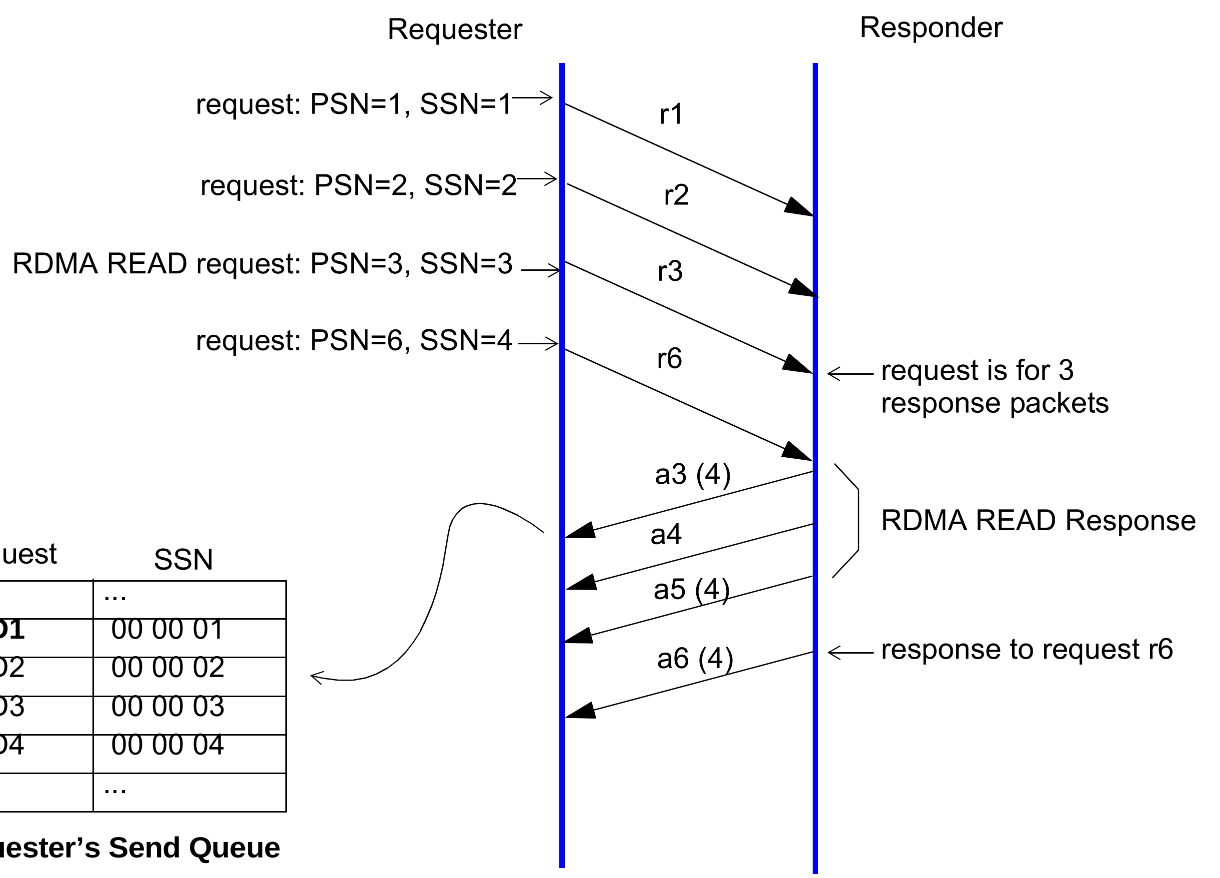
WQEs posted to its send queue. Since the responder may choose to coalesce acknowledges, a single response packet may in fact acknowledge several request messages. Thus, when it receives a new MSN, the requester begins evaluating WQEs on its send queue beginning with the oldest outstanding WQE and progressing forward. This is illustrated in the figure below for the case where there are no outstanding RDMA READ requests or ATOMIC Operation requests on the send queue.

Figure 99 Requester Behavior - Completing WQEs



'r' is a request packet.
'a' is an acknowledge packet.
MSN is shown in parentheses.

For the case where there are outstanding RDMA READ requests or ATOMIC Operation requests, the situation is slightly more complex. In this case, the requester only completes outstanding WQEs up to either the first outstanding RDMA READ request, ATOMIC Operation request, or WQE whose SSN matches the MSN in the response packet, whichever comes first. This is because both RDMA READ requests and ATOMIC



request	SSN
...	...
SEND1	00 00 01
SEND2	00 00 02
READ3	00 00 03
SEND4	00 00 04
...	...

Requester's Send Queue

Since RDMA READs are loosely ordered, it is likely that the responder will "complete" SEND4 before it finishes returning the READ response data (a3, a4, a5). Nonetheless, a3 has an MSN of 4 indicating that it the responder has completed SEND1, SEND2 and SEND4.

However, the requester may only complete SEND1 and SEND2 because of the presence of READ3 in the send queue.

'r' is a request packet
'a' is an acknowledge packet (message)
MSN is shown in parentheses.

\

Figure 100 Limitation on Completing Send Queue WQEs

Operation requests require an explicit response and thus cannot be completed until such an explicit response is received.

C9-149: For an HCA responder using Reliable Connection service, the MSN counter shall be inserted in the AETH regardless of whether the response is a positive acknowledgment, a negative acknowledgment or a duplicate acknowledgment.

o9-94: If a TCA responder implements Reliable Connection service, the MSN counter shall be inserted in the AETH regardless of whether the response is a positive acknowledgment, a negative acknowledgment or a duplicate acknowledgment.

9.7.7.2 END-TO-END (MESSAGE LEVEL) FLOW CONTROL

IBA provides an end-to-end (or message level) flow control capability for reliable connections that can be used by a responder to optimize the use of its receive resources. Essentially, a requester cannot send a request message unless it has appropriate credits to do so.

Encoded credits are transported from the responder to the requester in an acknowledge message in the Syndrome field of the AETH. The credits carried in the AETH are with respect to the MSN field of the same AETH; therefore proper interpretation of the credit field also requires interpretation of the MSN field. See Section 9.7.5.2 AETH Format on page 291 for a full description of the appropriate AETH fields.

Each credit represents the receive resources needed to receive one inbound request message. Specifically, each credit represents one WQE posted to the receive queue. The presence of a receive credit does not, however, necessarily mean that enough physical memory has been allocated. For example, it is still possible, even if sufficient credits are available, to encounter a condition where there is insufficient memory available to receive the entire inbound message.

1) The end-to-end credit mechanism applies only to Reliable Connected service.
2) End-to-End credits are generated by a responder's receive queue and consumed by a requester's send queue.
3) Requirements on a CA for supporting end-to-end flow control are given in Chapter 17: Channel Adapters on page 822. HCA receive queues must generate end-to-end credits, but TCA receive queues are not required to do so. If the TCA's receive queue generates End-to-End credits, then the corresponding send queue must receive and respond to those credits.
4) Credits are issued on a per message basis, without regard to the size of the message.
5) End-to-End credits are carried in the AETH as an encoded 5-bit field.
6) The responder may send credits to the requester asynchronously by using an Unsolicited acknowledge packet. An unsolicited acknowledge packet is created by re-sending the most recently sent acknowledge packet.

C9-150: Each HCA receive queue shall generate end-to-end flow control credits.

o9-95: Each TCA receive queue may generate end-to-end credits.

C9-151: If a TCA's given receive queue generates End-to-End credits, then the corresponding send queue shall receive and respond to those credits. This is a requirement on each send queue of a CA.

9.7.7.2.1 TRANSFERRING CREDITS FROM RESPONDER TO REQUESTER

Two mechanisms are defined for transporting credits from the responder's receive queue to the requester's send queue. The credits can be piggybacked onto an existing acknowledge message, or a special unsolicited acknowledge message can be generated by the responder. Piggybacked credits are those credits that are carried in the AETH field of an already scheduled acknowledge packet.

Piggybacked Credits:

Piggybacking of end-to-end credits refers to transferring credits to the requester in the AETH of a normal acknowledge packet. Credits are carried in AETH Syndrome[4:0]. Credits can be piggybacked onto any acknowledge packet when the MSN field in the AETH is also valid.

Unsolicited Acknowledge Packet:

From a PSN perspective, an unsolicited acknowledge message appears to the requester like a duplicate of the most recent positive acknowledge message. However, it always has an opcode of Acknowledge even if the most recent positive acknowledge was an RDMA READ Response or Atomic Response. Since the ACK/NAK protocol prohibits the responder from sending duplicate negative acknowledge packets (NAKs), an unsolicited acknowledge cannot be created by re-sending a NAK packet.

An unsolicited acknowledge may be sent by the responder at any time. The requester's send queue simply recovers the credit field and the MSN from the most recently received acknowledge packet.

Since an unsolicited acknowledge packet appears to the requester as a duplicate response, it has no effect on the requester other than the transfer of the credits.

C9-152: The MSN field of the unsolicited acknowledge packet must have a valid MSN field.

9.7.7.2.2 NEGOTIATING CONNECTIONS: INITIAL CREDITS

For each connection established, the use (or not) of end-to-end flow control is established separately for each direction. The capabilities of the receive queue determine the flow control characteristics for that half of the connection.

C9-153: If the receive queue signals that it is expecting to generate credits, then the corresponding send queue must observe the end-to-end flow control rules. If, on the other hand, the receive queue signals that it will not generate end-to-end flow control credits, then the corresponding send queue may transmit request messages at will without regard for credits. This is a requirement on each send queue of a CA.

C9-154: If a TCA's receive queue does not generate End-to-End credits, it shall place the value 5b11111 in AETH Syndrome[4:0] signalling that the credit field is invalid.

C9-155: When the receive queue is in the RESET state, the transport shall set the initial credit count to zero. Once the queue pair has transitioned to the INITIALIZED, RTR, SQD or RTS states, it shall increment its credit count for each receive WQE posted.

Once it is in the RTR, SQD or RTS states, the responder may transfer these credits to the requester by using unsolicited acknowledges.

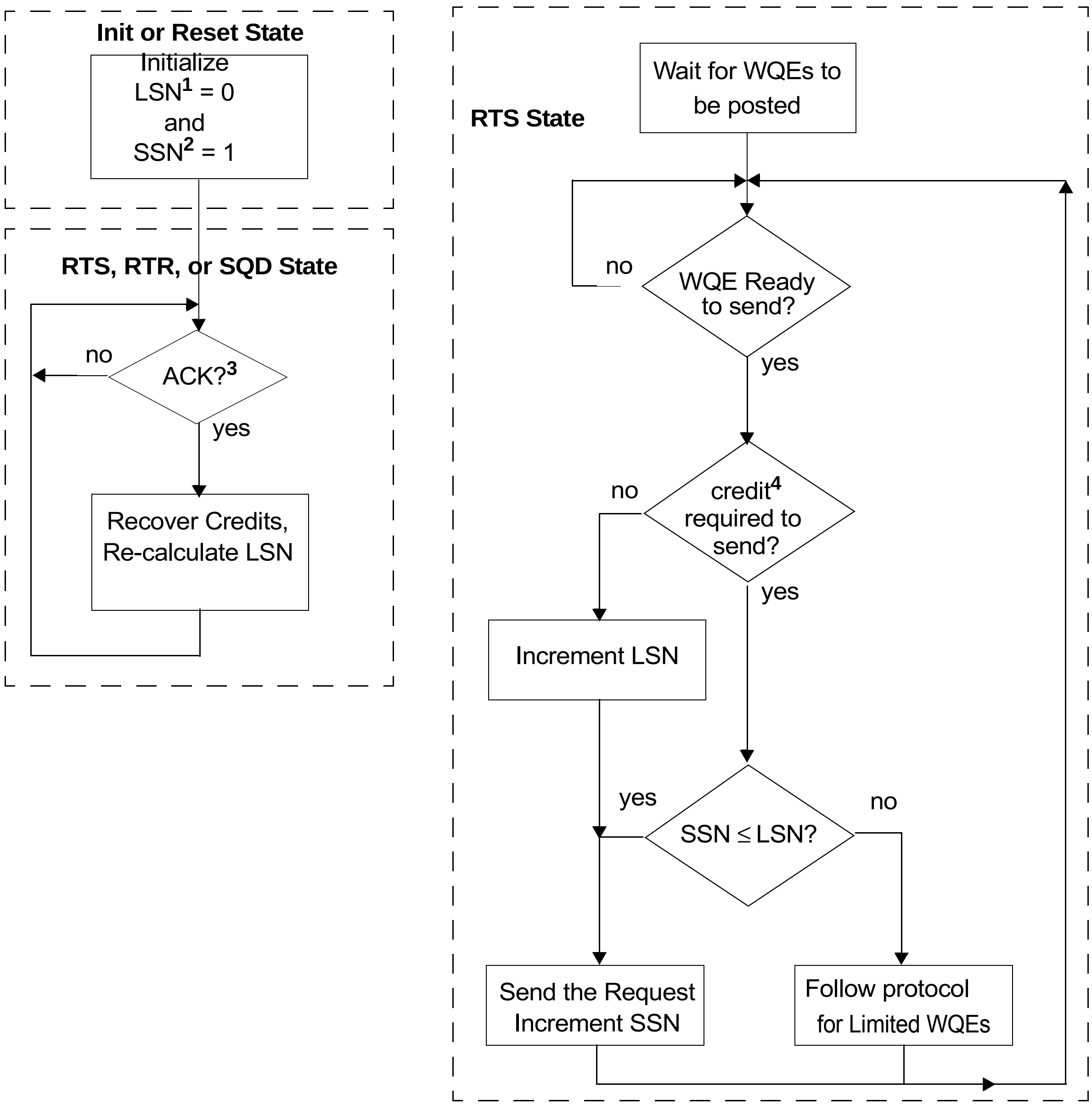
Normally an unsolicited acknowledge is created by re-sending the most recently sent positive acknowledge packet with an updated credit field. At initialization time however, no acknowledge packets have yet been sent so the normal method for creating an unsolicited acknowledge cannot be used. Therefore, at initialization time, an unsolicited acknowledge is created by subtracting “1” from the initial PSN. Thus, if the PSN is initialized to 0x000000 when the receive queue is in RESET state, then the PSN of the initial unsolicited acknowledge shall be 0xFFFFFF. “Initialization time”, in this context means the interval beginning when the receive queue has transitioned out of the RESET state and has not yet sent an acknowledge packet in either the RTR, SQD or RTS states.

To the send queue which receives this initial unsolicited acknowledge packet, it will appear as a “ghost” acknowledge packet Figure 96 Response Packet PSN Regions on page 300. The requester's send queue may accept the MSN and credits contained in the unsolicited acknowledge packet but ignore the rest of the packet. This is an exception to the normal rules for ghost responses which require that ghost acknowledge packets be dropped.

The above paragraph notwithstanding, responsibility for recovering initial credits from the responder shall lie with the requester; if the responder provides initial credits by using an unsolicited acknowledge, the requester may accept those as its initial credits in satisfaction of its responsibility to recover initial credits.

C9-156: If the responder does not provide initial credits, the requester shall behave as specified in Section 9.7.7.2.5 Requester Behavior - Limited Send WQEs on page 322

Figures Figure 101 Requester End-to-End Credit Processes on page 317 and Figure 102 Responder End-to-End Credit Initialization Process on page 318 describe this behavior. Note that these figures do not depict all normal state transitions for the receive and send queues. These are fully specified in the Software Interface chapter.



1) LSN = Limit Sequence Number. This is described below.
2) SSN = Send Sequence Number. This is described below.
3) Any ACK or unsolicited ACK with a valid MSN
4) a credit is required for a “Send” or “RDMA Write with immediate”.

Figure 101 Requester End-to-End Credit Processes

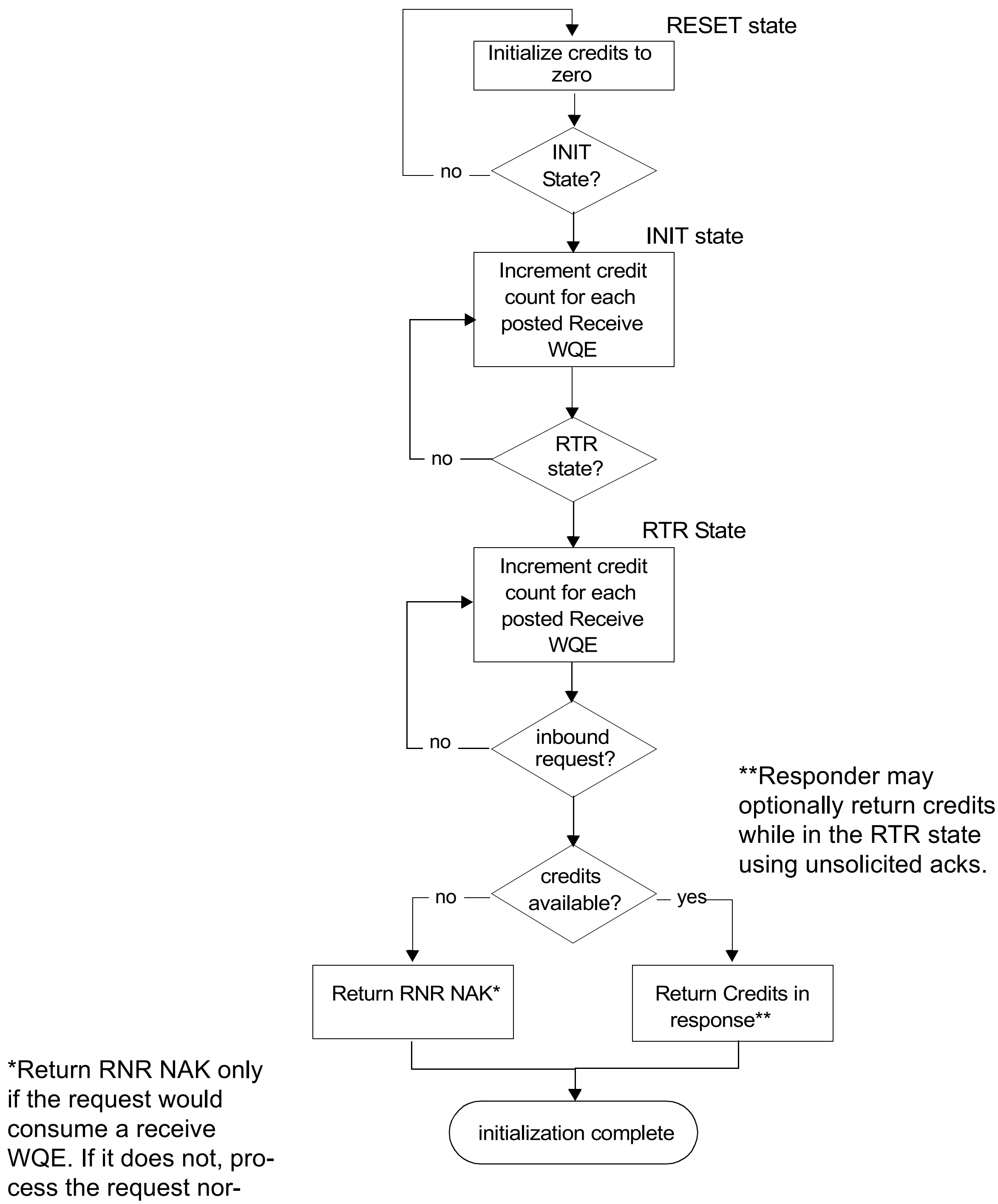


Figure 102 Responder End-to-End Credit Initialization Process

9.7.7.2.3 RESPONDER ALGORITHM FOR CALCULATING CREDITS

C9-157: For an HCA using Reliable Connection service, if the receive queue generates end-to-end flow control credits, it shall increment its credit count for each WQE posted to the receive queue. It shall decrement its credit count for each inbound request message received which consumes a WQE. Thus, the responder does not adjust its credit count when it receives an RDMA READ request, an RDMA WRITE request without Immediate data or an ATOMIC Operation request.

o9-96: If a TCA implements Reliable Connection service, and if the receive queue generates end-to-end flow control credits, it shall increment its credit count for each WQE posted to the receive queue. It shall decrement its credit count for each inbound request message received which consumes a WQE. Thus, the responder does not adjust its credit count when it receives an RDMA READ request, an RDMA WRITE request without Immediate data or an ATOMIC Operation request.

C9-158: For an HCA using Reliable Connection service, if the receive queue generates end-to-end flow control credits, for each acknowledge message generated, either a normal acknowledge message or an unsolicited acknowledge message, it shall insert its current encoded credit count as shown in Table 47 End-to-End Flow Control Credit Encoding on page 320, in AETH Syndrome[4:0]. For example, if the receive queue has five credits available, it shall insert the 5 bit value b00100 in the AETH. It also includes its current MSN value.

o9-97: If a TCA implements Reliable Connection service, and if the receive queue generates end-to-end flow control credits, for each acknowledge message generated, either a normal acknowledge message or an unsolicited acknowledge message, it shall insert its current encoded credit count as shown in Table 47 End-to-End Flow Control Credit Encoding on page 320, in AETH Syndrome[4:0]. For example, if the receive queue has five credits available, it shall insert the 5 bit value b00100 in the AETH. It also includes its current MSN value.

9.7.7.2.4 REQUESTER BEHAVIOR

The presence or absence of credits limits the sender's ability to transmit requests which will consume a receive WQE (SEND requests or RDMA WRITE requests with immediate data).

C9-159: The send queue's behavior when it has no credits available to it shall be as specified in Section 9.7.7.2.5 Requester Behavior - Limited Send WQEs on page 322.

The requester may always send a request which does not consume a receive WQE (RDMA WRITE request without immediate data, RDMA READ request, or ATOMIC Operation request) without regard to credits.

C9-160: The requester shall not violate the normal transaction ordering rules as stated throughout this specification, particularly in Section 9.5 Transaction Ordering on page 235.

In particular, the requester may not search the send queue looking for requests which don't consume a receive WQE and transmit those requests out of order, nor may the requester violate the rules governing fenced WQEs.

The available credits are encoded and carried in AETH Syndrome[4:0]; the MSN is carried in the least significant 3 bytes of the AETH. Table 47 below shows, for each valid encoded credit, the actual number of credits.

Table 47 End-to-End Flow Control Credit Encoding

Credit	Valued added to MSN to get LSN	Credit	Valued added to MSN to get LSN
00000	0	10000	256
00001	1	10001	384
00010	2	10010	512
00011	3	10011	768
00100	4	10100	1024
00101	6	10101	1536
00110	8	10110	2048
00111	12	10111	3072
01000	16	11000	4096
01001	24	11001	6144
01010	32	11010	8192
01011	48	11011	12288
01100	64	11100	16384
01101	96	11101	24576
01110	128	11110	32768
01111	192	11111	invalid

Logically, the requester associates a sequential Send Sequence Number (SSN) with each WQE posted to the send queue. The SSN bears a one-to-one relationship to the MSN returned by the responder in each response packet. Thus, the requester interprets the MSN as representing the SSN of the most recent request completed by the responder.

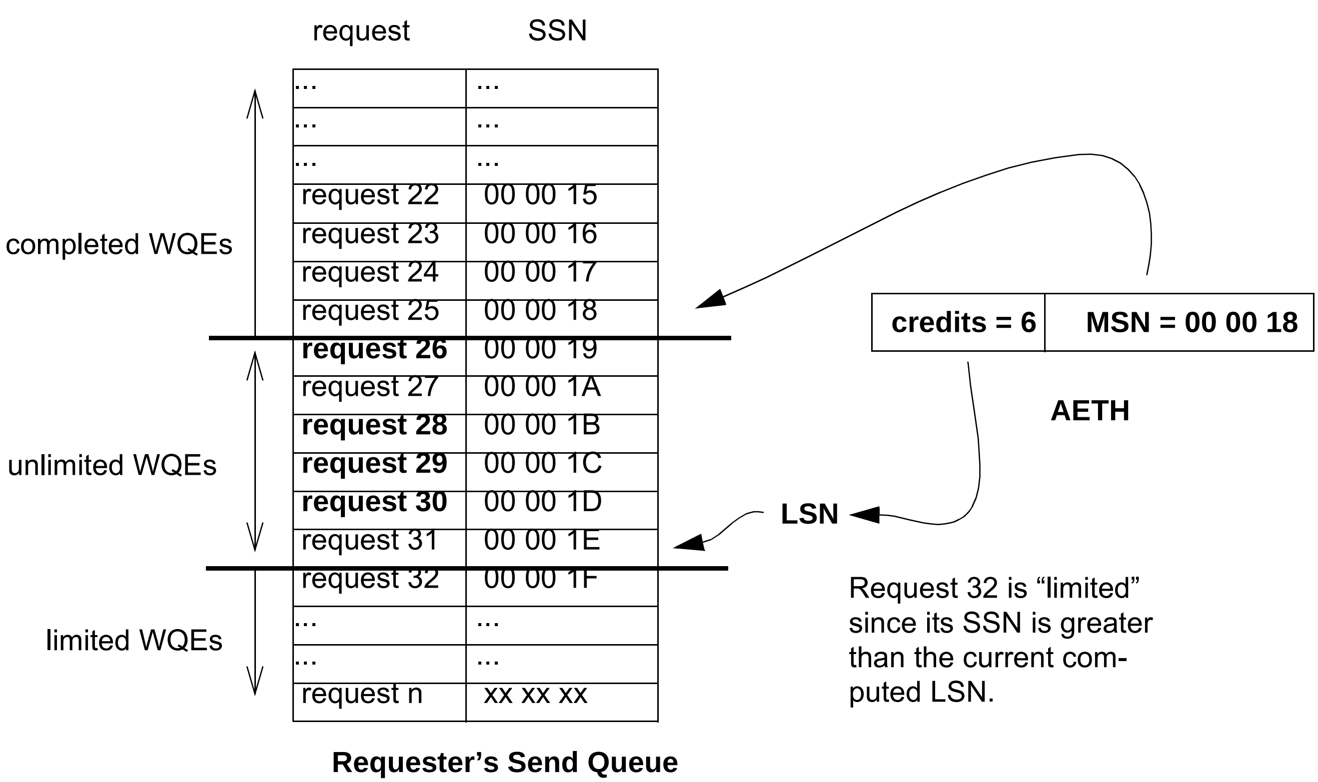
C9-161: The encoded credit count returned by the responder in the AETH shall specify the number of receive WQEs posted to the responder's receive queue relative to the MSN.

Since the MSN is directly related to the requester's SSN, the credit count is a simple offset into the send queue from the SSN of the most recent request completed by the responder. Logically, the sum of the MSN plus the credit count is the requester's Limit Sequence Number (LSN). The requester may freely transmit any request whose SSN is less than or equal to the computed LSN.

Any request whose SSN is greater than the current computed LSN is said to be limited. The send queue's behavior when it encounters a limited request is as specified in Section 9.7.7.2.5 Requester Behavior - Limited Send WQEs on page 322.

Figure 103 Relating AETH values to the Send Queue on page 321 illustrates the relationship between the values returned by the responder in the AETH and the requester's send queue.

Figure 103 Relating AETH values to the Send Queue



The requester calculates a new LSN each time it receives an acknowledge packet containing valid credits. The requester also dynamically adjusts the LSN by adding one to it for every request it wishes to send that does not consume a receive WQE (RDMA READ requests, RDMA

WRITE requests without immediate data, or ATOMIC Operation requests). This adjustment is the mechanism which allows the requester to send requests that do not consume a receive WQE.

Any given implementation is not required to implement the LSN and SSN mechanisms described above, but must conform semantically to the behavior described.

9.7.7.2.5 REQUESTER BEHAVIOR - LIMITED SEND WQES

C9-162: When the requester encounters a WQE on its send queue for which it has no available credits, that WQE is said to be limited. The send queue's behavior when it encounters a limited WQE shall be as follows:

- If the limited request WQE is an RDMA READ request, an RDMA WRITE request without immediate data, or an ATOMIC Operation request, it may be sent normally without regard to the availability of credits. The normal rules for ordering of requests still hold (i.e., the send queue may not search through the list of posted WQEs in an attempt to find unlimited WQEs to be sent out of order). After sending such a request, the requester increments its computed LSN value[1], since the sent request does not consume a receive WQE and thus does not consume a credit.
- If the limited request WQE is a SEND request, the send queue shall transmit no more than a single packet of the request message before it must stop transmission and wait for an acknowledge packet. To ensure that the responder will generate a response, the requester shall set the AckReq bit in that single packet.
- If the limited request WQE is an RDMA WRITE request with immediate data, the requester may transmit the entire request message before it must stop transmission and wait for an acknowledge packet. This is permitted because it is the single packet containing immediate data of the request that actually consumes the receive WQE. To ensure that the responder will generate a response, the requester shall set the AckReq bit in the last packet of the request message.

C9-163: For an HCA using Reliable Connection service, if the limited WQE is a SEND request, the send queue shall transmit no more than a single packet of the request message. Within this single packet, the Acknowledge Request (AckReq) bit of the BTH shall be set. The requester shall then stop transmission and wait for an acknowledge packet.

---

1. An interesting situation can occur that artificially limits the sender LSN with certain message patterns; if the sender does Send, RDMA, RDMA, RDMA with two credits from the receiver, it will increment the LSN by three. If after that, the response arrives with MSN+1 credit, the LSN will then be set back by two, putting the requestor into limit until the Ack from the RDMA's arrive.

o9-98: If a TCA implements Reliable Connection service, and if the limited WQE is a SEND request, the send queue shall transmit no more than a single packet of the request message. Within this single packet, the Acknowledge Request (AckReq) bit of the BTH shall be set. The requester shall then stop transmission and wait for an acknowledge packet.

o9-99: In an HCA using Reliable Connection service, or if a TCA implements Reliable Connection service, and if the limited request WQE is a RDMA WRITE request, the requester may transmit the entire request message before it must stop transmission and wait for an acknowledge packet. To ensure that the responder will generate a response, the requester shall set the Acknowledge Request (AckReq) bit in the last packet of the RDMA WRITE request.

C9-164: Since the responder's receive queue may generate an unsolicited acknowledge message at any time, the requester shall be prepared to receive an unsolicited acknowledge message from the responder at any time, provided that the receive queue has signalled that it will generate end-to-end flow control credits.

An unsolicited acknowledge is used solely for the purpose of transferring credits from the responder to the requester. On receiving an unsolicited acknowledge, the requester recalculates its LSN as specified above and responds accordingly.

A lack of credits does not impact a requester's ability to re-transmit previously transmitted requests as part of its recovery from lost packets. End-to-end credits only limit the transmission of new request messages. For example, if the requester detects a timeout condition after having sent a single packet of a limited SEND request, it decrements its timeout retry counter as usual and retransmits the request.

9.7.8 Reliable Datagram

Reliable Datagram provides reliable communication, i.e. the same level of reliability and error recovery as for Reliable Connection, using a one-to-many paradigm. A requestor's send queue may send sequential messages to different responders, at different QPs on the same or different nodes. A responder QP may receive messages from multiple requesters on the same or different endnodes. As with the Unreliable Datagram transport service, the source endnode and source QP are provided to the responder.

The motivation for using Reliable Datagram is to economize the QP name space for applications that engage in "all to all" communication. Consider N processor nodes, each with M processes. If all M processes wish to communicate with all the processes on all the nodes, a Reliable Connection service requires $M^2*(N-1)$ QPs on each node. By comparison, the Reliable Datagram service only requires M QPs + N “end-to-end” (EE) connections on each node for exactly the same communications.

Reliability is implemented using at least one “QP-like” context for each remote endnode - this is referred to as the End-to-End Context (EE Context or EEC). This context provides the information needed to locate the remote node, to serialize and exchange acknowledgments, and maintain reliability.

The Service still uses the QPs to provide the queues, WQE pointers, protection checking parameters etc. Together, a QP and an EEC contain the information needed to reliably move messages to a destination. But many QPs may use a single EEC for sending or receiving, and a QP may communicate through several different EECs, one chosen with each message.

When an application determines the target that it is to communicate with, it must first establish (or use an already established) an EE Context.

As with Reliable Connection, the local QPs to use for this service are established in the RD service mode by the application prior to use. Remote QPs are chosen in an application dependent manner.

Once the EE Context is created, the client may send a message to the responder QP via this EE Context. The client must specify the EE context handle (handle that defines the destination endnode), the responder QP, the Q_Key, and any additional message parameters. The implementation then “multiplexes” the messages from each source QP to the appropriate EEC, and sends the message. When the message arrives at the destination, the implementation uses the EE Context to validate the packet and “de-multiplexes” the message to the appropriate QP.

The Reliable Datagram service uses the methods (PSNs, ACK/NAK protocol etc.) as described previously in 9.7 Reliable Service on page 247.

9.7.8.1 RELIABLE DATAGRAM CHARACTERISTICS

Table 48 Reliable Datagram QP characteristics

Property / Level of Reliability	Support
Corrupt data detected	Yes
Data delivered exactly once (Except for an unrecoverable error; that is reported to the application)	Yes
Data order guaranteed to same destination QP from the same source QP	Yes
Data order guaranteed to different destinations from the same QP	Yes
Scalability (number of messages) on the service	Limited to number of EE Contexts in use between endnodes
Data loss detected	Yes
RDMA READ Support	Yes
RDMA WRITE Support	Yes
State of SEND/RDMA WRITE when request completed	Completion on remote end node
State of in-flight SEND/RDMA WRITE when unrecoverable error occurs	First one unknown, others not delivered
ATOMIC Support	Optional
Multi-packet message support	Yes
Multiple EE Context allowed between end-nodes (to provide traffic segregation for QOS)	Yes
Single SL / QoS assigned to EE Context	Yes
Number of messages in-flight per EEC	1
Number of messages in flight per QP	1
Number of messages enqueued per EEC / QP	Implementation limited only
Number of packets allowed in flight (architectural)	2^{23}
RD QP shall only communicate with RD QPs	Yes
Partition Key verification	On a per EEC basis
Protection verification (e.g. Q_Key, R_Key, etc.) and Addressing	On a per QP basis
Max Size of messages	2^{31}
Destination QP, Q_Key, and address supplied	On a per send WR basis
Source QP and address supplied	On a per receiver completion basis

o9-100: CA’s claiming to support RD mode shall provide QP’s capable of supporting RD. When operating in RD mode, these QPs allow sending sequential RD messages to different responders, at different destination QPs on the same or different nodes. When operating in RD mode, these

QPs shall be capable of receiving RD messages from multiple requesters on the same or different endnodes.

o9-101: CA's claiming to support RD mode shall provide EEC's that allow the "multiplexing" of multi packet RD message traffic to and from multiple QPs while maintaining reliability (messages are delivered from a requester to a responder at most once, in order and without corruption, or the upper layer is notified.)

o9-102: CA's claiming to support RD mode shall ensure that an RD message has been completed at the sender (fully acknowledged or completed in error) before sending another message on the same EEC.

o9-103: CA's claiming to support RD mode shall ensure that an RD message has been completed at the sender (fully acknowledged or completed in error) before sending another message on the same QP.

o9-104: CA's claiming to support RD mode shall meet the requirements specified in 9.2.1 Operation Code (OpCode) on page 206 for coding of the RD OpCodes, 9.3.1 Reliable Datagram Extended Transport Header (RDETH) - 4 Bytes on page 210 for creation of that header, and 9.6 Packet Transport Header Validation on page 236 and 9.7 Reliable Service on page 247 through 9.7.6 for reliable transports for processing RD messages.

o9-105: CA's claiming to support RD mode shall meet the requirements specified in 9.9 Error detection and handling on page 362 while processing RD messages.

o9-106: CA's claiming to support RD mode shall provide support for setting up connections between EECs as defined in Chapter 12: Communication Management on page 545, using the Management facilities as defined in Chapter 13: Management Model on page 595

o9-107: CA's claiming to support RD mode shall ensure that RD message errors or events that are not associated with the underlying EE Context (for example Q_Key or R_Key violations or RNR-NAK) shall not cause that EE Context to shut down or prevent the EE Context from processing other RD messages destined to other QPs.

o9-108: HCA's claiming to support RD mode shall provide support for Send, RDMA WRITE, RDMA READ, and ATOMICS in RD mode to the extent defined and reported in 11.2 Transport Resource Management on page 476.

o9-109: HCA's claiming to support RD mode shall provide support for EEC management as defined in 10.2.6 End-to-End Contexts on page 405 and 11.2.6 EE Context on page 503.

o9-110: HCA's claiming to support RD mode shall provide support for RDD domains as defined in 10.2.7 Reliable Datagram Domains on page 406, 11.2.1.7 Allocate Reliable Datagram Domain on page 482, and 11.2.1.8 Deallocate Reliable Datagram Domain on page 483.

Implementation note: For many implementations, an EEC will actually be a special "mode" of a general QP or EE context. For these implementations, the context number specified as a destination EEC must be set up in Reliable Datagram 'EEC' mode. Reliable Datagram packets arriving at a context (identified by the EE Context field in the header) that is not set up to "EE Context" mode, shall be silently dropped.

The responder QP context must be set to support Reliable Datagram transport service. If a Reliable Datagram packet arrives at a QP context that is not configured for RD operation, the responder shall respond with a "NAK Invalid RD Request".

An important distinction for this service is that errors that are not associated with the underlying EE Context do not result in shutting that EE Context down. Examples of these would be Q_Key or R_Key violations. Similarly, the Receiver Not Ready (RNR NAK), caused by resources associated with the receiver's QP does not prevent the EE Context from processing other messages destined to other QPs.

Errors that are associated with the EE Context (retry limit exceeded, etc.), detected during a message transmission or reception, shall be reported in the WR completion.

Errors associated with the requestor or responder QP shall be reported in the WR completion with the usual error semantics. See 9.9 Error detection and handling on page 362 for a more complete discussion on errors.

Since an end-to-end credit mechanism is not practical in a "connectionless" type of service, responders shall send a NAK Receiver Not Ready response if a requester's SEND arrives while the responder's Receive Queue is empty. See 9.7.5.2.8 RNR NAK on page 295 for additional details.

To preserve the ordering rules required of this service, and to keep the design complexity down, messages on this service are sent one at a time from the source QP with the requirement that each message acknowledgment be received at the requesting QP before the next message can be started.

9.7.8.2 EXAMPLE RD OPERATIONS

The following is not normative material, but is included to clarify this topic. These examples are based on an HCA implementation; other implemen-

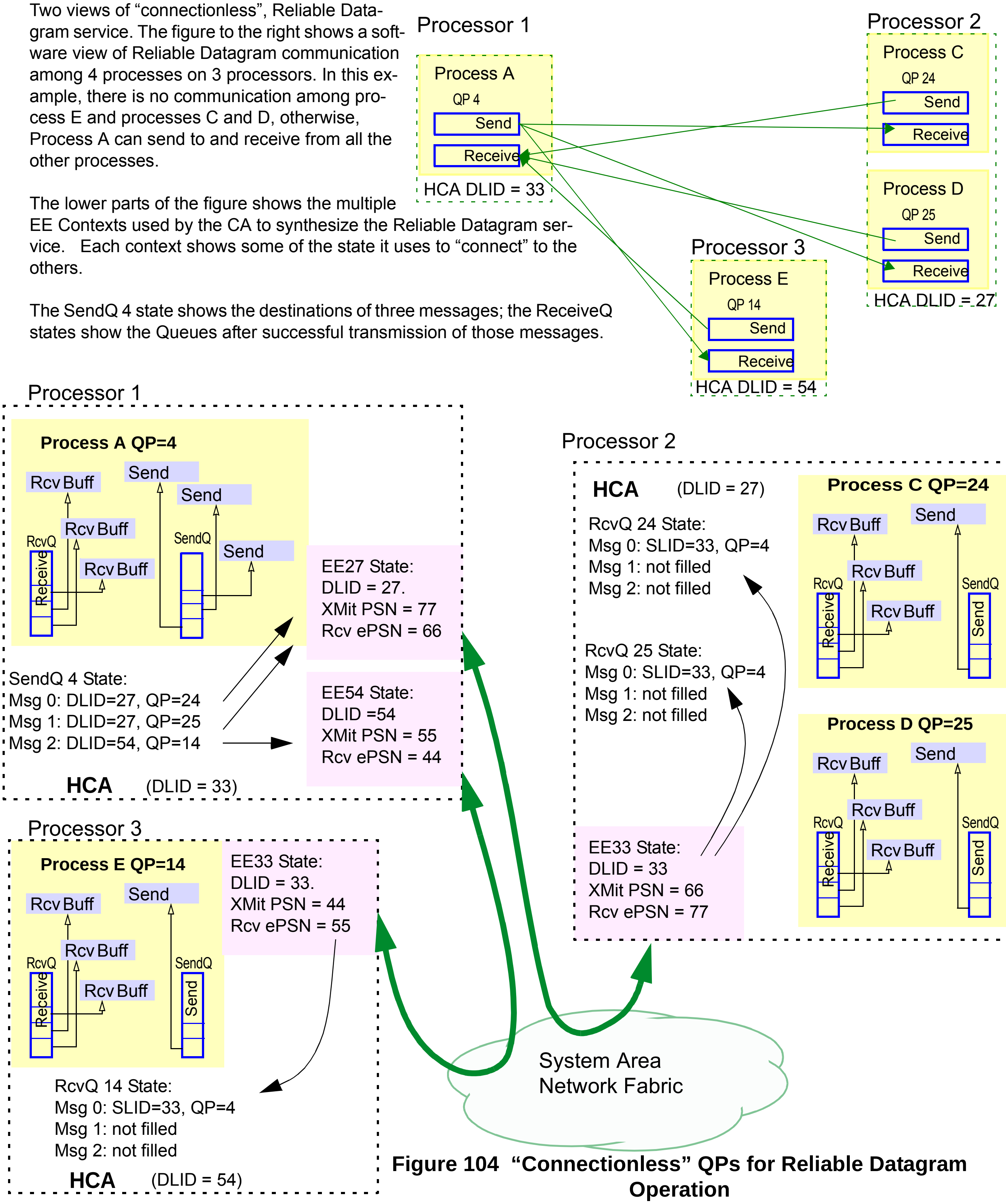


Figure 104 "Connectionless" QPs for Reliable Datagram Operation

tations are possible. TCAs, for instance may not utilize virtual memory and may modify other details of this example.

This implementation example maintains a standard send queue for WRs.

It also maintains a linked list of Send QPs, anchored at each EEC. This list contains QPs, each of which has a WR at the head of its Send Queue that is destined for the EEC.

In order to manage the orderly transmission of packets and messages, the implementation uses a scheduler. This scheduler maintains a list of those EEC that have packets to send. As each EEC gets to the head of the scheduler list, one or more packets are sent (depending on QoS and other factors not important here).

On the responder side, the implementation example maintains a standard set of Receive Queues.

The implementation also maintains space in each EEC to copy those parameters needed to process a single incoming message. These parameters are copied both from the QP (PD, CQ, Q_Key etc.) and the receive WQE (data segment L_Key, Virtual address, size etc.). The EEC then has enough information to process the entire message to completion with no further reference to the WQE or QP, even if the message contains many packets.

9.7.8.2.1 EXAMPLE OUTBOUND REQUEST

1) The client of the Reliable Datagram posts a send message (described by WQE) to the send queue of its QP. This consists of:
   - the list of data segments (virtual address, L_Key and length) that describes the send message
   - the destination “EE Context number”
   - the destination QP number.
   - the destination Q_Key
2) When the WQE reaches the head of the Send Queue (found by pointer from the QP context), the EE Context is located from the WQE and the QP is “linked” to the EECs “QP list” for processing (EEC contains enqueue and dequeue pointers, each QP contains link pointer to next QP to run. Take QP at enqueue pointer, update its “next” link to point to the new QP, adjust enqueue pointer to the newly linked QP).

   If the EEC is not currently sending messages, the EEC is also placed into the scheduler.
3) When the EEC is scheduled to send a message, the HCA locates the WQE parameters by accessing the QP at the head of the EEC’s “QP

list" (Take QP found at the Dequeue pointer) and using the QP's work queue pointers.

4) HW uses the memory protection parameters of the enqueuing process (stored with the QP Context) and the virtual address etc. from the WQE. This allows the Send Queue HW to directly access the virtual address space of each process that posts send message buffers.

5) The HCA hardware reads the data buffer, builds the transport header (including the "Packet Sequence" number associated with the EE Context) and puts the packet onto the wire.

6) This process is repeated from step 3 until the entire message is sent. The "EE Context" is serviced according to the same scheduling algorithm used for Reliable Connection QPs.

7) When a message is completely sent, the CA waits until all Acknowledgments are in for the message.

8) Since the EEC must wait for a message ACK before continuing (only a single message outstanding at once), the EEC is scheduled with an appropriate timeout and the EE Context is updated.

9) When the last ACK has arrived and the WQE completed, the HCA determines if there are additional WQEs posted to the current QP (the one at the head of the EEC's QP list). If so, the next WQE is examined to locate the EEC for the QP's next message (this may be to a different EEC than the current). The CA then dequeues the QP from the current EEC's QP list, and enqueues it on the tail of the next message's EEC QP list. This is similar to step 2 above.

10) The EEC's QP list is examined to determine if any QP has work for this EEC.

11) The process repeats from step 3 until no more messages are available to send. At this point, the EEC is removed from the scheduler and set into an "inactive" state.

9.7.8.2.2 EXAMPLE INBOUND REQUEST

The inbound request needs to access the QP state associated with the responder's Receive Queue, the receive WQE, and the EE Context that maintains information about the source. Both QP and EEC are available in the header for this purpose.

The following lists the steps taken by the HCA to process an incoming request packet:

First or Only Packets

1) The incoming request packet arrives and is found to be un-corrupted and the first or only packet of the message.

2) The packet header specifies the destination QP number. This is the QP associated with the client of the Reliable Datagram service. This

QP points to the receive queue, and a WQE, but does not have any sequence number information. The packet header also includes the "EE Context number" that is used to access the EE Context. The sequence number information is stored with the EE Context connected to the requesting host.

3) The incoming request's sequence number is compared against the state of the EE Context connected to the requesting node.

4) If the sequence number and other packet contents are correct, the destination QP's memory protection and WQE entry information are temporarily copied to the EE Context. This implementation is useful because other EECs may be targeting the same QP and other messages will end up in progress to the same QP. By copying the WQE and memory related information to the EEC, the QP is free to point to subsequent WQEs for additional messages. This is also the reason that receive WQEs may complete out of order.

5) The memory protection checks are done, and if the receive buffer is valid, the incoming request is written to memory (or in the case of a RDMA READ, stored for later processing).

6) The CA puts the EEC on the scheduler to send an ACK response.

7) If the packet was an "only", then the CA completes the message using the EEC's copy of WQE and QP values, with no additional references to the QP or WQE.

Middle or Last packets

1) For subsequent packets from the same message, only the EE Context is accessed based on the header EEC number. This allows other EECs to utilize other WQEs from the QP Receive Queue independently.

2) If the sequence number and other header checks are correct, the memory protection checks are done, and if the receive buffer is valid, the incoming request data is written to memory.

3) The CA puts the EEC on the scheduler to send an ACK response.

4) If the packet was a "last", then the CA completes the message using the EEC's copy of WQE and QP values, with no additional references to the QP or WQE.

9.7.8.2.3 EXAMPLE OUTBOUND ACKNOWLEDGE

When the EEC gets to the head of the scheduler queue, the CA notes that an ACK must be sent, and sends it. If multiple packets have arrived before the EEC gets to the head of the scheduler, this creates a coalesced ACK. The EE Context's last valid receive sequence number is sent in the ACK packet per the ACK/NAK rules.

If the operation was an RDMA read, then multiple response packets may be required. In this case, the EEC is placed back on the scheduler after each packet until the PSN of the responses reaches the expected PSN.

9.7.8.2.4 EXAMPLE INBOUND ACKNOWLEDGE

A returning ACK response indicates a request packet was successfully completed. When the ACK arrives, the EE Context is examined and the returned PSN is checked. If this is the expected (next sequential) ACK, the expected PSN is updated. If this is the last ACK of a message and all previous packets were acknowledged, then the message can be completed using the EEC's copy of the QP and WQE information. If the ACK is not sequential, then the usual coalesced ACK rules apply. Since only a single message is outstanding at one time, only a single message is ever acknowledged at one time.

For RDMA READs, the CA uses the EEC's copied QP protection information and WQE data segment information to store the data.

9.7.8.3 RELIABLE DATAGRAM OPERATIONS

The processing is very much the same as defined for Reliable connection service. The significant difference is for the treatment of repeated packets at the responder, and the rules for repeating a request at the requester. The differences are highlighted in *italics*.

9.7.8.3.1 SEND AND RDMA WRITE WITH IMMEDIATE DATA PROCESSING

SENDs and RDMA WRITEs with Immediate data are handled in the same way as for Reliable Connection service, *except that end-to-end credits are not returned to the sending QP.*

o9-111: CA's claiming support for Reliable datagram service shall use the NAK-RNR protocol to indicate an over-run of the Receive Queue for RD messages.

9.7.8.3.2 RDMA READ PROCESSING

RDMA READs are handled in the same way as for Reliable Connection service. Incoming requests are stored at the responder's "hidden resources", attached to the EE Context, and memory protection information is accessed or copied from the QP Contexts. *Unlike Reliable Connection service, the number of RDMA READ request messages outstanding from a single QP or EEC shall be limited to one*.

9.7.8.3.3 ATOMICS PROCESSING

Atomics are handled in the same way as for Reliable Connection service. Incoming requests are stored at the responder's "hidden resources", attached to the EE Context, and memory protection information is accessed or copied from the QP Contexts. *Unlike Reliable Connection service, the number of ATOMIC requests outstanding from a single QP or EEC shall be limited to one*.

9.7.8.4 ORDERING RULES

Receive Queues are FIFO queues. Once enqueued, WQEs shall begin processing in FIFO order, but *may be completed out of order*. The messages from any single source QP shall always be in order.

o9-112: CA's claiming to support RD mode shall provide upper layer support for out of order receive queue completion for RD messages.

Send queues are FIFO queues. Once enqueued, WQEs shall be processed for sending in the order they were enqueued.

o9-113: CA's claiming to support RD mode shall ensure that WQEs on the Send Queue in RD mode are completed in order whether they are targeting different destination QPs on the same or a different endnode or the same destination QP. The completions for WQEs shall always be returned to the transport consumer in FIFO order.

This does not mean that the implementation must place the data portions of the messages in memory in any particular order. As a result, the arrival order is not guaranteed until the message is marked complete on at least one side. An application shall expect that memory buffers are undefined until the message is completed.

Note that items queued on different QP's Send Queues on the same HCA for the same destination endnode or even the same destination QP are not ordered with respect to each other. For example, if WQE 'A' destined for destination'X' and QP "75" is posted to QP 1, and WQE 'B' destined for destination'X' and QP "75" is later posted to QP 2 of the same CA, there is no guarantee that 'A' will arrive before 'B' at the destination.

o9-114: For CA's claiming to support RD mode, upper layers must tolerate lack of ordering among RD messages from different send QPs. That is, items queued on different QP's Send Queues on the same HCA for the same destination endnode or even the same destination QP are not ordered with respect to each other.

9.7.8.5 HANDLING QP ERRORS - RESYNC

Since RD service allows multiple QPs to share a single EEC, it is desirable that a QP with an error localized to the QP, not effect the remainder of the QPs sharing the same EEC. To support this, RD service allows the EEC to "Abandon" or "Suspend" operations on a QP under certain error conditions.

A message is "Abandoned" if it is completed in error at the source QP, and the QPn is transitioned to the error state.

A message is “Suspended” if it is not completed at the source QP, but another message is started on the same EEC. When later resumed, the suspended message must be restarted from the beginning if it is a Send, or RDMA Write with immediate operation. If it is an RDMA Read or RDMA Write w/o Immediate, it can (implementation choice) be restarted where it left off, but must appear to be a new message to the responder.

“Suspend” and Restart only apply to RNR NAK conditions, where the responder is temporarily unable to perform the request associated with a particular QP, and the requestor can improve performance by sending messages from other QPs on the same EEC, while waiting for the RNR NAK timeout.

A requestor is not allowed to Suspend an Atomic operation.

For convenience, we will sometimes say a message is “aborted” when when either “abandoned” or “suspended” is meant.

The concept of “Abandon” or “Suspend” does require the RD service to deal with a class of errors that can occur when packets associated with a message are delayed, repeated, or both. To deal with this problem, a RESYNC operation is required whenever a message is “Abandoned” or “Suspended”. An example of this problem is shown in Figure 105 below where an RNR’d request is actually executed by the responder.

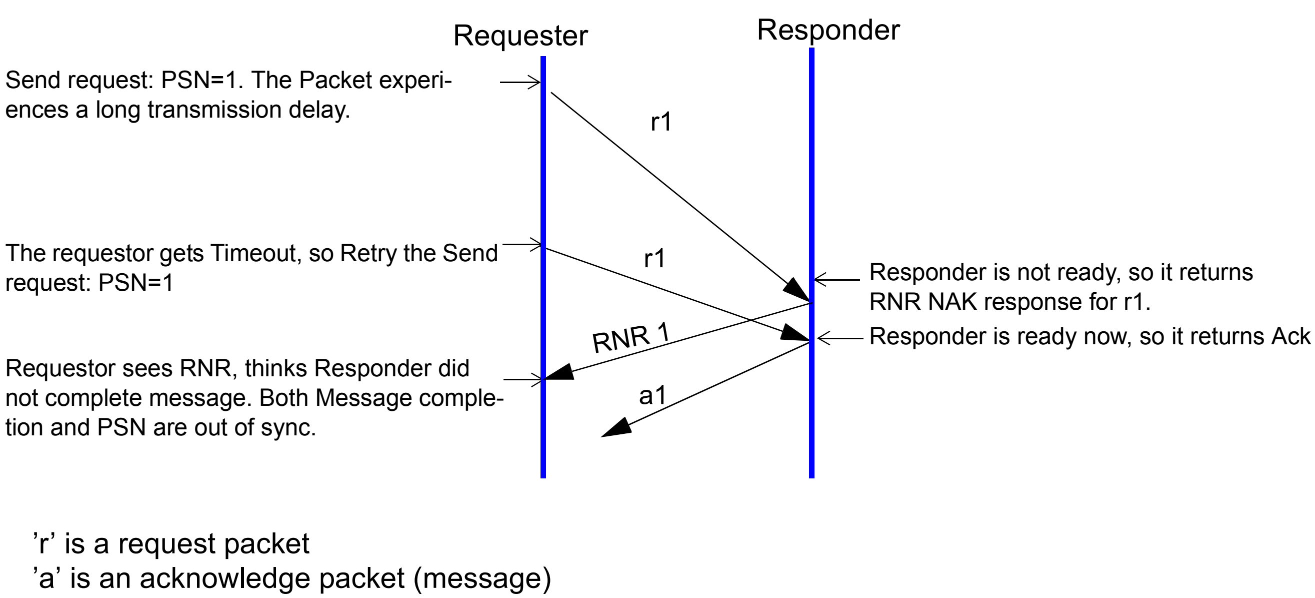


Figure 105 Loss of synchronization of the EEC on Suspend

If the requestor simply started a new message at the original PSN, data corruption could occur. In addition, the requestor, when it restarted the RNR NAK'd message later, the responder would get two copies.

To deal with this problem, a RESYNC is used following any error that affects a QP, but leaves the EEC able to continue operation. The RESYNC serves several purposes, it gets the PSNs on both ends of the EEC synchronized, it lets the responder know that the previous message, if incomplete, is to be abandoned, and it allows the requestor to determine with certainty the status of the current message at the responder. The following flow chart shows the resynchronization process at the requestor.

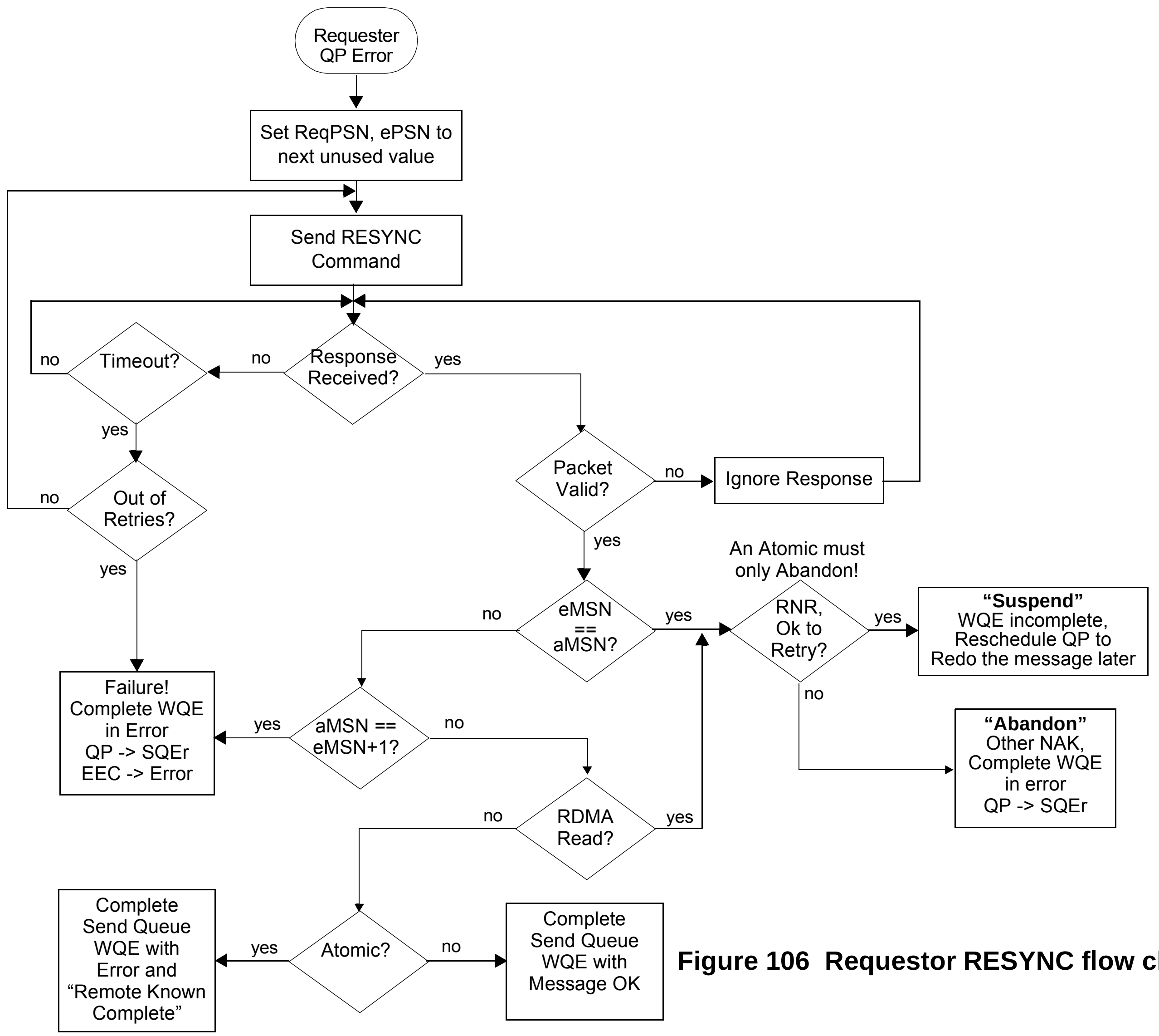


Figure 106 Requestor RESYNC flow chart

The following ladder diagram illustrates a RESYNC operation to correct the problem described in Figure 105. The RESYNC process allows getting both ends to agree on a appropriate PSN, to determine the status of the aborted message at the responder, and to inform the responder to abort a message in progress, if that is the case.

.

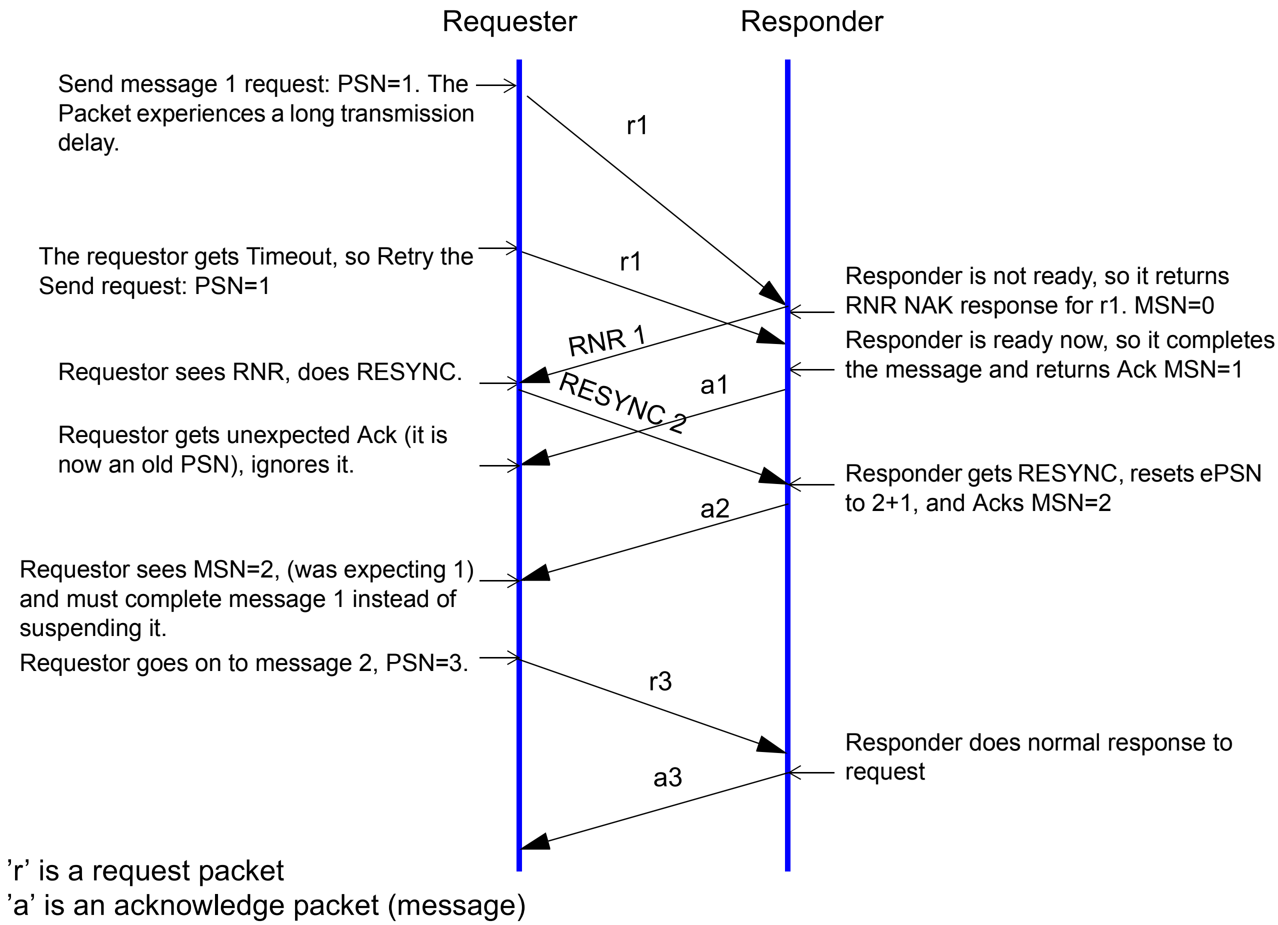


Figure 107 RESYNC detects unexpectedly complete message

Another problem that RESYNC corrects is dealing with “Ghost” packets. This is illustrated in where a multi-packet message has an error on an early packet, the requestor puts the Send Queue in SQEr, and the upper layer eventually returns the Send Queue to RTS. Meanwhile, a fabric “event” causes a packet to be extremely delayed.

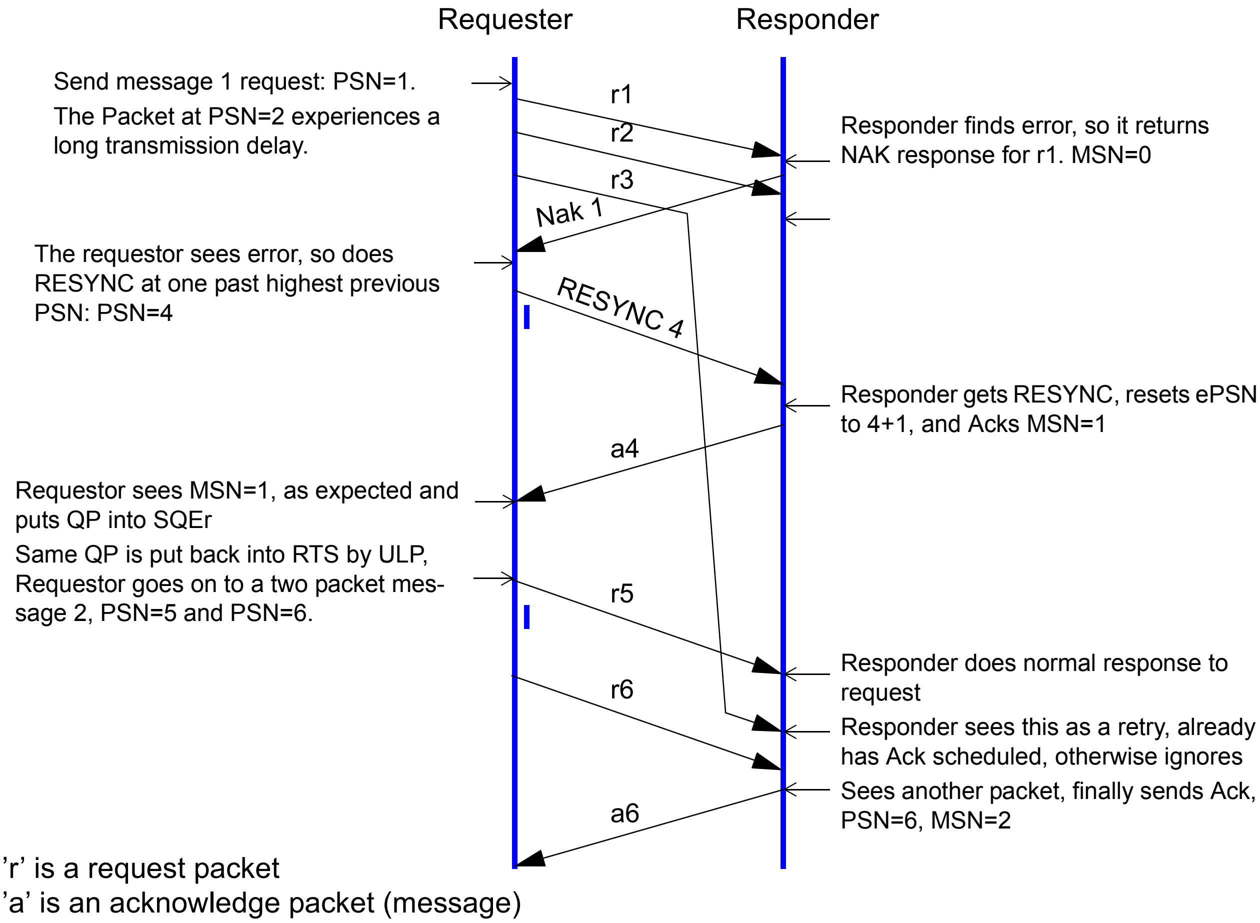


Figure 108 RESYNC prevents corruption by delayed packets

In the figure above,’r3’, being extremely delayed, arrives at the responder during packet processing of a message for the same QPs. In this case, the responder must see’r3’ as a retried packet. As such it is ignored, except to schedule an Ack, at least for Sends and RDMA Writes. For RDMA reads, the responder must create an explicit response. If a correct RDMA read was already in progress, it must be interrupted and a different response generated. This will create ghost responses at the requestor, and cause it to timeout on its RDMA Read request, with a retry to make a recovery. For an atomic, the response also appears as a ghost, or unexpected response at the requestor. In either case, it is dropped and the operation recovers.

If the RESYNC had not been performed, message 2 would have started with PSN=2. The reception of'r3' at PSN=3 would have created a data corruption problem.

o9-114.a1: For CA's claiming to support RD mode, when a message in RD mode incurs a QP related error the requestor may either:

1) Transition the QP and EEC to the error state and complete the message in error, or
2) Implement the RESYNC process as described in 9.7.8.5 Handling QP errors - RESYNC on page 333.

The RESYNC request generation is described in 9.7.3.2.2 RESYNC Generation on page 258.

o9-114.a2: For CA's that support the RD transport service, following the sending of a RESYNC, the requestor shall wait for an Ack at the RESYNC PSN. No other response is valid.

As usual, the AckReq bit should be set to insure that the responder schedules a response.

RESYNC should be timed out and retried with the usual retry count if no correct response arrives as described in 9.7.6.1.3 Detecting Lost Acknowledge Messages and Timeouts on page 305.

As usual, the requestor updates the request PSN by one prior to sending another message from the EEC.

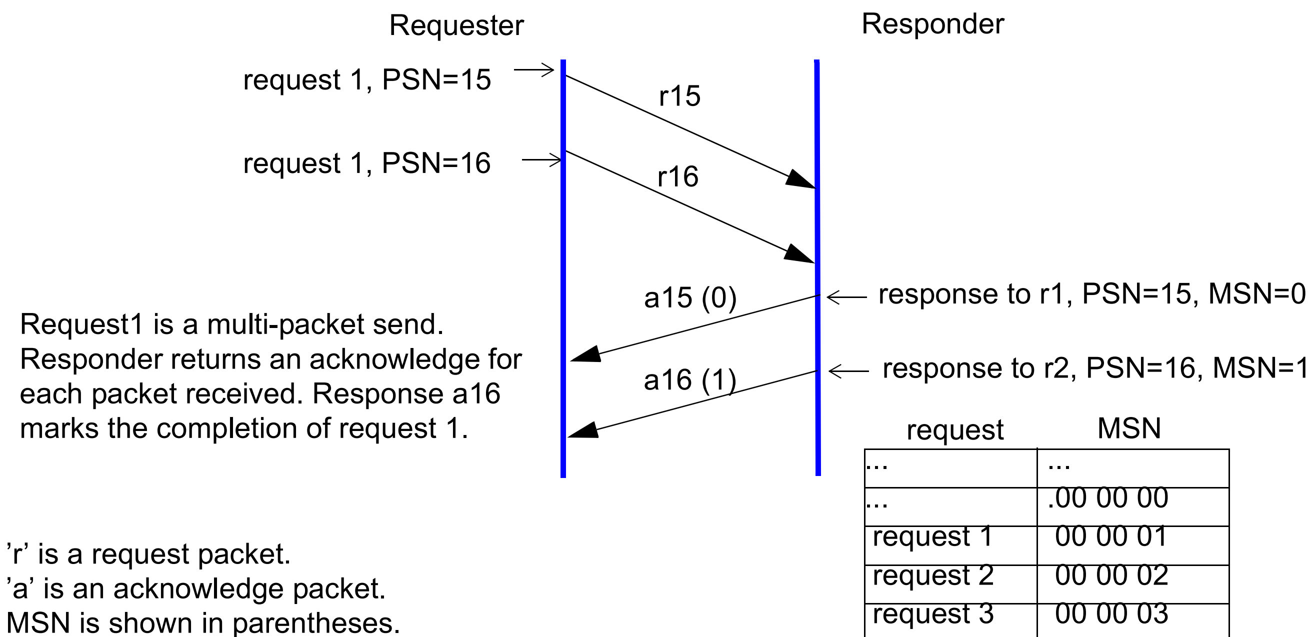
The RESYNC response may actually complete two messages, the previous message, and the "RESYNC" message.

9.7.8.6 RESPONDER GENERATION OF MSN

For Reliable Datagram service, the Message Sequence Number is a number returned by the responder to the requester indicating the number of messages completed by the responder at the EE context. The MSN is carried in the three least significant bytes of the AETH. The MSN assists the requester in completing WQEs by informing the requester of the messages that have been completed by the responder.

o9-114.a3: The Responder in CAs that implement Reliable Datagram service shall return an MSN in the AETH of every response packet.

o9-114.a4: A CA responder using Reliable Datagram service shall initialize its MSN value to zero. The responder shall increment its MSN whenever it has successfully completed processing a new, valid request message. The MSN shall not be incremented for duplicate requests. The incremented MSN shall be returned in the last or only packet of an RDMA READ or Atomic response.



request	MSN
...	...
...	.00 00 00
request 1	00 00 01
request 2	00 00 02
request 3	00 00 03
...	...

Figure 109 Responder use of MSN

Responder's MSN

9.7.8.6.1 REQUESTER BEHAVIOR ON RECEIVING A NEW MSN

The existence of a new MSN value in a response packet may be used by the requester as a signal to complete a message.

The MSN in the response to the "RESYNC" message may also be used to determine the completion status of the previous message:

- If, due to earlier retries, the previous message was actually completed, the MSN will be two higher than the last value, which was returned in the NAK response.
  - If the previous message was a Send or RDMA Write, it should be completed normally (meaning that it was not abandoned or suspended after all).
  - If the previous message was an RDMA Read, the requestor did not get the return data, so must still either suspend or abandon the message, depending on the error type.

    - If the previous message was an Atomic, it must be completed in Error, the additional information “Known Complete at Responder” may be returned to the upper layers.
- If the MSN was as expected (one greater than last value), the previous message should be treated appropriately:
    - Completed in error if it is abandoned, the additional information that the message was “Known incomplete at Responder”
    - Not started if it is a Send, or RDMA Write with immediate, to be retried later
    - Incomplete if it is an RDMA Read or RDMA Write without immediate to be restarted as a new message later

If the MSN was any other value, the response is corrupted, and the EEC should be put into the error state.

9.8 UNRELIABLE SERVICE

IBA defines two types of unreliable service: Unreliable Connection (SEND, RDMA WRITE) and Unreliable Datagram (SEND only). These services have the following characteristics:

1) Requester receives no acknowledgment of message receipt
2) No packet order guarantees
3) Responder validates incoming packets as normal (validates appropriate header fields, CRC checks). A corrupted packet may be silently dropped, causing the message to be dropped.
4) On detecting an error in an incoming packet such as a dropped / out of order packet, the responder does not stop, but continues to receive incoming packets.
5) Responder considers the operation complete once it has received a complete message in correct sequence, all data has been committed to the local fault zone, and all appropriate validity checks (including variant and invariant CRC checks) have been completed. For Unreliable Connected service, the definition for a completed message is given in section 9.8.2.2.7 on page 355
6) Requester considers a message operation complete once the “last” or “only” packet has been committed to the fabric.

9.8.1 VALIDATING AND EXECUTING REQUESTS

This section applies to both unreliable connection and unreliable datagram services. Where there are differences between the services, those differences are noted. The major differences between the two services are due to the fact that Unreliable Datagram service is restricted to single packet messages whereas Unreliable Connected service does not have this restriction. In addition, Unreliable Datagram service is restricted to using the Send function further simplifying the request validation process.

The following describes the requirements placed on a responder for validating an inbound request packet.

C9-165: The responder shall validate the various fields of the headers in order to verify the integrity of the packet. This validation process is specified in Section 9.6 Packet Transport Header Validation on page 236. Packets containing invalid fields shall be silently dropped by the responder.

C9-166: For an HCA using Unreliable Connected service, the PSN shall be examined to detect out of order packets. By examining the PSN, the responder can determine whether the packet is a new request or an invalid packet. See Section 9.8.2.2.1 Responder - Validating the PSN on page 348 for a description of this check.

o9-115: If a TCA implements Unreliable Connected service, the PSN shall be examined to detect out of order packets. By examining the PSN, the responder can determine whether the packet is a new request or an invalid packet. See Section 9.8.2.2.1 Responder - Validating the PSN on page 348 for a description of this check.

C9-167: For an HCA using Unreliable Connected service, the responder shall examine the packet OpCode to determine that the packet OpCode sequence is valid. This check is not applicable to Unreliable Datagram since that service is restricted to single packet messages, thus the concept of a sequence of opcodes is not applicable.

o9-116: If a TCA implements Unreliable Connected service, the responder shall examine the packet OpCode to determine that the packet OpCode sequence is valid. This check is not applicable to Unreliable Datagram since that service is restricted to single packet messages, thus the concept of a sequence of opcodes is not applicable.

C9-168: The responder shall examine the packet OpCode to determine whether the requested operation is supported by this receive queue.

C9-169: The responder shall verify that it has sufficient resources available to receive the message. The necessary resources include a valid receive WQE (for a SEND or an RDMA Write with immediate data), and, for a SEND request, sufficient buffer space available to receive the request.

C9-170: For an HCA responder using Unreliable Connection service, if the request is for an RDMA WRITE operation, the responder shall examine the R_Key. If the packet is found to be valid, in order, and sufficient resources are available, it is executed by the responder. In the process of execution, the responder may encounter local errors.

o9-117: If a TCA responder implements Unreliable Connection service, and if it supports RDMA operations, it shall behave as follows. If an inbound request is for an RDMA WRITE operation, the responder shall examine the R_Key. If the packet is found to be valid, in order, and sufficient resources are available, it is executed by the responder. In the process of execution, the responder may encounter local errors.

C9-171: For an HCA responder using Unreliable Connection or Unreliable Datagram services, or for a TCA responder using Unreliable Datagram service, the responder shall follow the sequence shown in Figure 110 when validating an inbound request packet.

o9-118: If a TCA responder implements Unreliable Connection service, the responder shall follow the sequence shown in Figure 110 when validating an inbound request packet.

For Unreliable Connected service, these requirements are discussed in some detail in section 9.8.2.2 Responder Behavior on page 348. Packet validation for Unreliable Datagram service is discussed in 9.8.3.2 Responder Behavior on page 358.

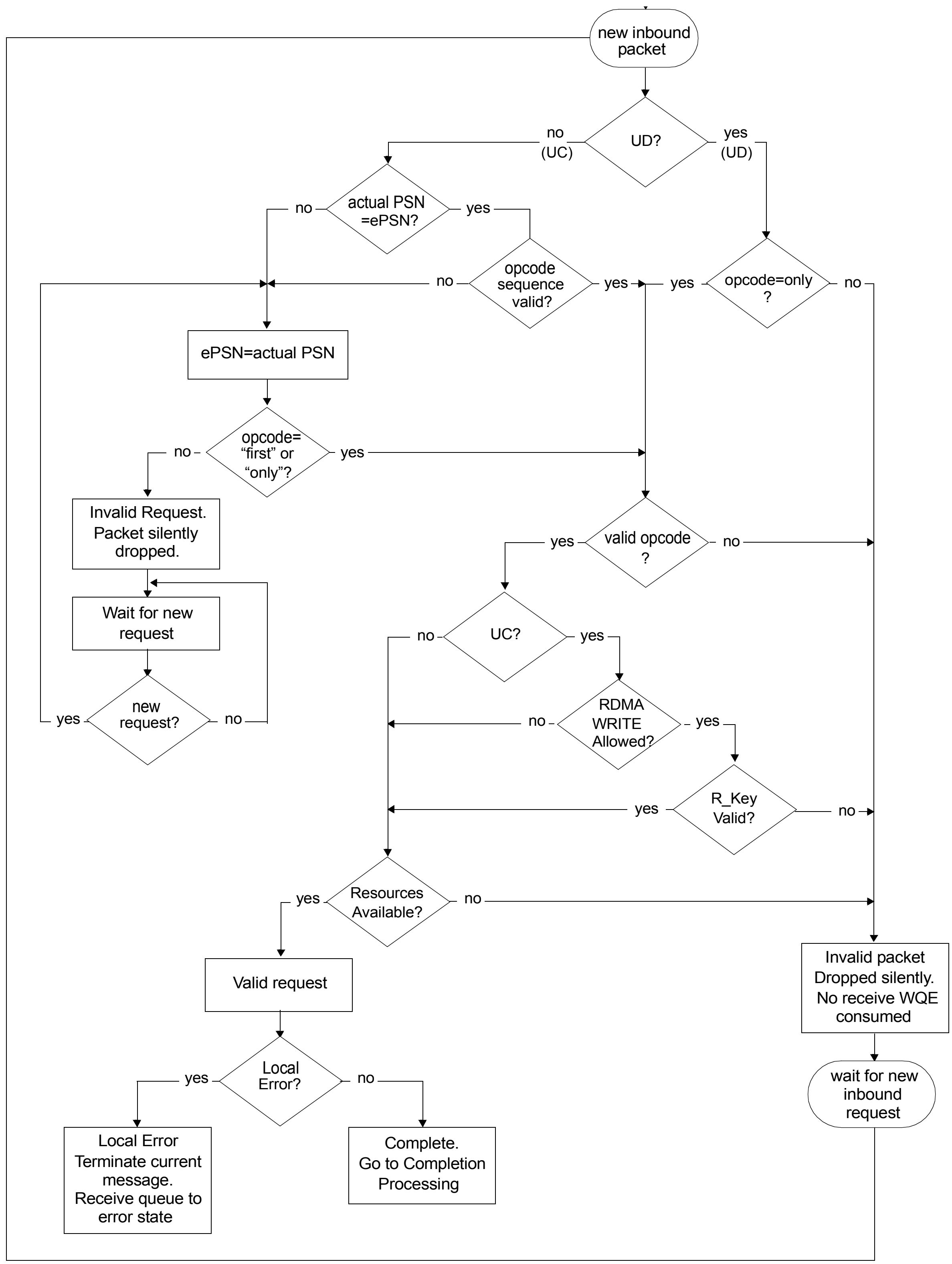


Figure 110 Unreliable Service: Inbound Packet Validation

OM10531

9.8.2 UNRELIABLE CONNECTIONS

An unreliable connection consists of a one-to-one correspondence between two QPs. Packets are sent from one QP to the other but no acknowledgments are generated by the destination QP. The chief characteristics are that there are no delivery guarantees made to the requester. The responder, however can detect data corruption and out of order packets.

The characteristics of Unreliable Connection service are summarized in Table 49.

Table 49 Summary of Unreliable Connection Service Characteristics

Characteristic	Comment
Delivery guarantee	No guarantees to the requester. Responder may drop messages.
Ordering-requester	No guarantee. Requester cannot rely on msgs arriving in order.
Ordering-responder	Responder detects and drops out of order packets.
Ordering-responder	Dropped packets may cause the message to be dropped.
Ordering-responder	After dropping a packet, responder resumes with the first packet of a new message.
Supported Operations	Sends and RDMA WRITEs (with and without Immediate data)
Message size	Maximum 2^{31} bytes. Msgs may comprise multiple packets.

9.8.2.1 REQUESTER BEHAVIOR

This section specifies the requester's required behavior when generating request packets for Unreliable Connection service.

9.8.2.1.1 REQUESTER - GENERATING PSN

C9-172: For an HCA requester using Unreliable Connection service, the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

o9-119: If a TCA requester implements Unreliable Connection service, the requester must place a value, called the current PSN, in the BTH:PSN field of every request packet.

During connection establishment, the transport layer's client must program the next PSN to any value between zero and 16,777,215.

C9-173: For an HCA requester using Unreliable Connection service, the initial PSN, as programmed by the transport layer's client, shall appear as the BTH:PSN in the first request packet generated by the requester.

o9-120: If a TCA implements Unreliable Connection service, the initial PSN, as programmed by the transport layer's client, shall appear as the BTH:PSN in the first request packet generated by the requester.

C9-174: For an HCA using Unreliable Connection service, the transport layer shall modify (update) the PSN only when the send queue is in a proper state to transmit request packets. For example, for an HCA, the transport layer does not update the next PSN while the queue pair is in the INITIALIZED state.

o9-121: If a TCA implements Unreliable Connection service, the transport layer shall modify (update) the PSN only when the send queue is in a proper state to transmit request packets.

C9-175: For an HCA using Unreliable Connection service, each request packet generated by the requester must have a PSN value that is an increment of "1" (modulo $2^{24)}$ of the PSN value of the preceding request packet.

o9-122: If a TCA implements Unreliable Connection service, each request packet generated by the requester must have a PSN value that is an increment of "1" (modulo $2^{24)}$ of the PSN value of the preceding request packet.

Table 50 Requester's Calculation of Next PSN

Current Request Packet	PSN for Next Request Packet
SEND, RDMA WRITE	current PSN + 1 (modulo 2^{24})

9.8.2.1.2 REQUESTER - GENERATING OPCODES

The opcodes generated by a requester must fit into a schedule of opcodes as shown below.

C9-176: For an HCA requester using Unreliable Connection service, the requester must generate packet opcodes which fit within the schedule of valid OpCode sequences as shown in Table 51 Schedule of Valid OpCode Sequences on page 347. When generating a request packet, the BTH:Opcode shall be as specified in Table 35 OpCode field on page 207.

Table 51 Schedule of Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

o9-123: If a TCA requester implements Unreliable Connection service, the requester must generate packet opcodes which fit within the schedule of valid OpCode sequences as shown in Table 51 Schedule of Valid OpCode Sequences on page 347. When generating a request packet, the BTH:Opcode shall be as specified in Table 35 OpCode field on page 207.

9.8.2.1.3 REQUESTER - GENERATING PAYLOADS

The requester shall generate payload lengths as a function of the opcode as follows:

C9-177: For an HCA using Unreliable Connection service, if the OpCode specifies a "first" or "middle" packet, then the packet payload length must be a full PMTU size.

C9-178: For an HCA using Unreliable Connection service, if the OpCode specifies a "only" packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

C9-179: For an HCA using Unreliable Connection service, if the OpCode specifies a "last" packet, then the packet payload length must be between one and PMTU bytes in size.

o9-124: If a TCA implements Unreliable Connection service, then it shall conform to the three preceding HCA requirements for OpCode.

9.8.2.1.4 COMPLETING A MESSAGE SEND OR RDMA WRITE

C9-180: For an HCA requester using Unreliable Connection service, the requester shall consider a message Send (or RDMA WRITE) complete when either of the following conditions occurs: The requester has committed the last byte of the VCRC field of the last packet to the wire (and detected no local errors associated with the message transfer), or the requester has detected a local error associated with the message transfer that causes the requester to terminate sending the request.

o9-125: If a TCA requester implements Unreliable Connection service, the requester shall consider a message Send (or RDMA WRITE) complete when either of the following conditions occurs: The requester has committed the last byte of the VCRC field of the last packet to the wire (and detected no local errors associated with the message transfer), or the requester has detected a local error associated with the message transfer that causes the requester to terminate sending the request.

Note that at the time that the requester completes the send WQE, the state of the memory at the responder is unknown. Likewise, if the requester detects a local error while sending the request packet, the state of the responder's memory is unknown.

9.8.2.2 RESPONDER BEHAVIOR

This section specifies the responder's required behavior when receiving inbound requests.

9.8.2.2.1 RESPONDER - VALIDATING THE PSN

The responder maintains an Expected PSN value (ePSN) that it uses to detect missing packets from a multi-packet request message and to detect dropped messages. Since the PSN of every inbound request packet is sequential and monotonically increasing for UC service, a break in the PSN sequence indicates a lost or dropped request packet.

C9-181: For an HCA responder using Unreliable Connection service, the responder shall maintain an Expected PSN value (ePSN). This is the PSN that the responder expects to find in the BTH of the next inbound request packet.

o9-126: If a TCA responder implements Unreliable Connection service, the responder shall maintain an Expected PSN value (ePSN). This is the PSN that the responder expects to find in the BTH of the next inbound request packet.

The responder's expected PSN may be initialized at connection establishment time by the transport's client to any value between zero and 16,777,215. However, since the responder will accept any valid packet with an opcode of "first" or "only", and use the value of the PSN contained in such a packet as its expected PSN, it is not required that the responder's initial expected PSN be programmed. See Chapter (Chapter 12: Communication Management on page 545 for a full description of the mechanism for loading the expected PSN at connection establishment time.

The initial expected PSN can only be set by the client when the queue is in the Initialized state. Attempts by the client to set the PSN when it is in any other state may be ignored by the transport layer.

C9-182: For an HCA using Unreliable Connection service, the transport layer shall modify (update) its expected PSN only when the receive queue is in a proper state to receive inbound request packets. For example, for an HCA, the transport layer does not modify the PSN when the queue pair is in the Initialized state.

o9-127: If a TCA implements Unreliable Connection service, the transport layer shall modify (update) its expected PSN only when the receive queue is in a proper state to receive inbound request packets. For example, for an HCA, the transport layer does not modify the PSN when the queue pair is in the Initialized state.

C9-183: For an HCA responder using Unreliable Connection service, an inbound request packet shall be declared out of order if its PSN does not exactly match the responder's current ePSN.

o9-128: If a TCA responder implements Unreliable Connection service, an inbound request packet shall be declared out of order if its PSN does not exactly match the responder's current ePSN.

C9-184: An HCA responder using Unreliable Connection service shall behave as follows. If, during packet validation, an inbound request packet is discovered with an OpCode of "first" or "only", the responder shall accept the packet and shall accept the PSN of that request message as its new ePSN, regardless of whether the inbound packet is out of order or not. This shall be done regardless of the previous value of ePSN.

o9-129: A TCA responder implementing Unreliable Connection service shall behave as follows. If, during packet validation, an inbound request packet is discovered with an OpCode of "first" or "only", the responder shall accept the packet and shall accept the PSN of that request message as its new ePSN, regardless of whether the inbound packet is out of order or not. This shall be done regardless of the previous value of ePSN.

C9-185: For an HCA responder using Unreliable Connection service, before executing an inbound request, the responder shall check the PSN by comparing the PSN in the inbound BTH to the responder's expected PSN. The rules that the responder uses to calculate its next expected PSN shall be the same as those used by the requester when it calculates the PSN value to insert in its next request packet. These rules are given in 9.8.2.1.1 Requester - Generating PSN on page 345.

o9-130: For an HCA responder using Unreliable Connection service, before executing an inbound request, the responder shall check the PSN by comparing the PSN in the inbound BTH to the responder's expected PSN. The rules that the responder uses to calculate its next expected PSN shall be the same as those used by the requester when it calculates the PSN value to insert in its next request packet. These rules are given in 9.8.2.1.1 Requester - Generating PSN on page 345.

o9-131: If the PSN of the inbound message does not match the responder's ePSN, the responder may notify its client of the presence of one or more lost messages. The mechanism by which the responder notifies its client is outside the scope of this specification.

C9-186: For an HCA responder using Unreliable Connection service, if a multi-packet message is in progress at the time that an out of order packet is detected, the current message shall be silently dropped. The responder then waits for the first packet of a new message. It is possible that the present packet (the out of order packet) is the first packet of a new message. If so, it shall be treated as a new message.

o9-132: If a TCA responder implements Unreliable Connection service, if a multi-packet message is in progress at the time that an out of order packet is detected, the current message shall be silently dropped. The responder then waits for the first packet of a new message. It is possible that the present packet (the out of order packet) is the first packet of a new message. If so, it shall be treated as a new message.

A "new message" is denoted by an inbound request packet with an OpCode in the BTH of "first" or "only".

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, excluding the present packet.

9.8.2.2.2 RESPONDER - OPCODE SEQUENCE CHECK

A request packet must fit within a schedule of valid OpCode sequences. The OpCode sequence is determined by examining the BTH:OpCode.

C9-187: For an HCA responder using Unreliable Connection service, the responder shall check the sequence of packet OpCodes as described in items (1) through (5) below:

1) If this is the first packet following establishment of the connection, then the packet OpCode must indicate either “first” or “only”. An OpCode of “middle” or “last” implies that at least the first packet of the current message was lost and denotes an invalid OpCode sequence.
2) If the last valid packet received had an OpCode indicating “first”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (SEND, RDMA WRITE). A current OpCode of “first” or “only” implies that at least the last packet of the previous message was lost and denotes an invalid OpCode sequence.
3) If the last valid packet received had an OpCode indicating “middle”, then the current OpCode must indicate either “middle” or “last”. It must also match the operation type specified in the last valid packet (SEND or RDMA WRITE request). A current OpCode of “first” or “only” implies that at least the last packet of the previous message was lost and denotes an invalid OpCode sequence.
4) If the last valid packet received had an OpCode indicating “last”, then the current OpCode must indicate either “first” or “only”. A current OpCode of “middle” or “last” implies that at least the first packet of the current message was lost and denotes an invalid OpCode sequence.
5) If the last valid packet received had an OpCode indicating “only”, then the current OpCode must indicate either “first” or “only”. A current OpCode of either “middle” or “last” implies that the first packet of the current message was missed and denotes an invalid OpCode sequence.

o9-133: If a TCA responder implements Unreliable Connection service, the responder shall check the sequence of packet OpCodes as described in items (1) through (5) above.

The responder’s behavior in the presence of an invalid OpCode sequence is specified in Section 9.9.3 Responder Side Behavior on page 375.

C9-188: For an HCA responder using Unreliable Connection service, if the responder detects an invalid OpCode sequence, the current message shall be silently dropped. The responder then waits for a new inbound request packet with an OpCode of “first” or “only”; any other inbound request packet shall be silently dropped.

o9-134: If a TCA responder implements Unreliable Connection service, and if the responder detects an invalid OpCode sequence, the current message shall be silently dropped. The responder then waits for a new inbound request packet with an OpCode of “first” or “only”; any other inbound request packet shall be silently dropped.

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, excluding the present packet.

C9-189: For an HCA responder using Unreliable Connection service, if the present packet, which caused the invalid OpCode sequence, has an OpCode of "first" or "only" it shall be treated as the first packet of a new request message.

o9-135: If a TCA responder implements Unreliable Connection service, and if the present packet, which caused the invalid OpCode sequence, has an OpCode of "first" or "only" it shall be treated as the first packet of a new request message.

The list of valid OpCode sequences is summarized in the following table.

Table 52 Summary: Valid OpCode Sequences

Previous Packet OpCode	Valid OpCodes for Current Packet
None e.g., first packet following connection establishment	"First" packet "Only" packet
"First" packet	"Middle" packet (message is 3 or more packets) "Last" packet (message is exactly 2 packets) Type of operation must match the previous OpCode
"Middle" packet	"Middle" packet "Last" packet Type of operation must match the previous OpCode
"Last" packet	"First" packet (1st packet of a new message) "Only" packet (1st packet of a new single packet msg)
"Only" packet	"First" packet "Only" packet

9.8.2.2.3 RESPONDER OPCODE VALIDATION

C9-190: For UC, the responder shall validate the requested function (SEND or RDMA WRITE) is supported by the receive queue and that the BTH:OpCode is not reserved before executing the request.

Note that the OpCode was also examined as part of packet validation in section 9.6 Packet Transport Header Validation on page 236 to ensure that the inbound packet contains a request for Unreliable Connected service.

C9-191: Invalid UC requests shall be silently dropped by the responder per 9.9.3 Responder Side Behavior on page 375.

9.8.2.2.4 RESPONDER REMOTE ACCESS VALIDATION

C9-192: For an HCA responder using Unreliable Connection service, if the inbound request is for a RDMA WRITE and the requested DMA length in the RETH is non-zero, then the following conditions shall be checked:

- The R_Key field in the RETH is valid,
- The virtual address and length specified in the RETH are within the locally defined limits associated with the R_Key,
- The type of access specified (Write) is within the locally defined limits associated with the R_Key.

A failure of any of these checks constitutes an R_Key violation. The responder's behavior in response to an R_Key violation is specified in Section 9.9.3 Responder Side Behavior on page 375.

o9-136: If a TCA responder implements Unreliable Connection service and RDMA functionality, it shall conform to the preceding HCA compliance statement.

C9-193: For an HCA using Unreliable Connection service, the R_Key field shall not be checked for a zero-length RDMA WRITE request, even if the request includes Immediate data.

o9-137: If a TCA responder implements Unreliable Connection service and RDMA functionality, the R_Key field shall not be checked for a zero-length RDMA WRITE request, even if the request includes Immediate data.

9.8.2.2.5 RESPONDER - LENGTH VALIDATION

C9-194: For an HCA responder using Unreliable Connection service, the PktLen field of the LRH shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE. This check applies only to SENDs.

o9-138: If a TCA responder implements Unreliable Connection service, the PktLen field of the LRH shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE. This check applies only to SENDs.

The length of the packet shall also be validated by comparing it to the OpCode as follows:

C9-195: For an HCA responder using Unreliable Connection service, if the UC BTH:OpCode specifies a "first" or "middle" packet, then the packet payload length must be a full PMTU size.

o9-139: If a TCA responder implements Unreliable Connection service, and if the UC BTH:OpCode specifies a “first” or “middle” packet, then the packet payload length must be a full PMTU size.

C9-196: For an HCA responder using Unreliable Connection service, if the UC BTH:OpCode specifies a “only” packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

o9-140: If a TCA responder implements Unreliable Connection service, and if the UC BTH:OpCode specifies a “only” packet, then the packet payload length must be between zero and PMTU bytes in size. Thus, the only way to create a zero byte length transfer is by use of a single packet message.

C9-197: For an HCA responder using Unreliable Connection service, if the UC BTH:OpCode specifies a “last” packet, then the packet payload length must be between one and PMTU bytes in size.

o9-141: If a TCA responder implements Unreliable Connection service, and if the UC BTH:OpCode specifies a “last” packet, then the packet payload length must be between one and PMTU bytes in size.

C9-198: For an HCA responder using Unreliable Connection service, if the request is an RDMA WRITE, the total amount of payload data received shall be compared to the DMA Length field specified in the RETH.

o9-142: If a TCA responder implements Unreliable Connection service and RDMA functionality, and if the request is an RDMA WRITE, the total amount of payload data received shall be compared to the DMA Length field specified in the RETH.

C9-199: For an HCA responder using Unreliable Connection service, if the BTH:OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA WRITE Middle), the pad count bits are verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, the OpCode is invalid.

o9-143: If a TCA responder implements Unreliable Connection service, and if the BTH:OpCode field[4:0] specifies a first or middle request packet (e.g. SEND First, or RDMA WRITE Middle), the pad count bits are verified to be b00, indicating no pad bytes are present. If the pad count bits are non-zero, the OpCode is invalid.

If a packet is detected with an invalid length, or the total amount of RDMA WRITE data does not match the DMA Length field in the RETH, the request is an invalid request. The responder’s behavior in such a case is specified in Section 9.9.3.1 Responder Side Error Response on page 377.

9.8.2.2.6 RESPONDER - LOCAL OPERATION VALIDATION

A valid inbound request may still fail to complete due to a failure that is local to the responder, e.g. local memory translation error while accessing local memory. A local error may cause the receive queue to transition to the error state. See 9.9.3 Responder Side Behavior on page 375 for additional details.

9.8.2.2.7 COMPLETING A MESSAGE RECEIVE

The responder considers a given inbound message completed successfully when it has:

- Detected the beginning of a valid message as indicated by the presence of a "First packet" or "Only packet" OpCode in the BTH,
- Detected the end of the same valid message as indicated by the presence of a "Only packet" or "Last packet OpCode in the BTH, without a skip in the PSN sequence,
- Received all the packets between "First packet" and "Last packet" inclusive successfully and in order, or has successfully received an "Only packet".
- Committed the message payload to the local fault zone without error, and,
- Successfully completed all appropriate validity checks (including variant and invariant CRC).

A failure detected during any of these steps may or may not cause the associated WQE to be completed in error. In some cases, such as a missing "first" packet, it is entirely likely that no WQE will be consumed by the responder. Note that, in the presence of errors, it is not possible to guarantee the state of the responder's memory. Some or all of a given packet may have been committed to the responder's memory before the error is detected.

Once an inbound message receive is completed successfully, the responder completes the current WQE.

9.8.3 UNRELIABLE DATAGRAMS

Unreliable Datagrams are a form of communication that allow a source QP to send each message to one of many destination QPs that may exist on the same or multiple destination endnodes.

- For each message to be sent, the requester must be supplied with the destination address (see 11.2.2.1 Create Address Handle on page 483), the destination QP, the destination Q_Key etc. See 11.4.1.1 Post Send Request on page 525 for the parameters supplied for an HCA.

- The responder must deliver to the client the requester's address, QP etc. See 11.4.2.1 Poll for Completion on page 531 for more detail on HCA requirements.

Table 53 Unreliable Datagram QP characteristics

Property / Level of Reliability	Support
Corrupt data detected and dropped	Yes, silent drop on error
OpCode Service and command Validation	Yes, silent drop on error
Receive buffer overrun	Yes, reported as WR error
Data repeated	No
Data order guaranteed	No
Data loss detected	Not required
RDMA Support	No
ATOMIC Support	No
Immediate data support	Yes
Max Size of SEND messages	PMTU-sized packet - 256 - 4096 bytes of data payload. Any message that exceeds the PMTU will not be delivered.
State of SEND when request completed	Committed to transmission on the fabric

C9-200: Devices that source UD messages shall limit the UD message size to a single packet. The packet should be no larger than the PMTU between the source and destination (or it will be dropped).

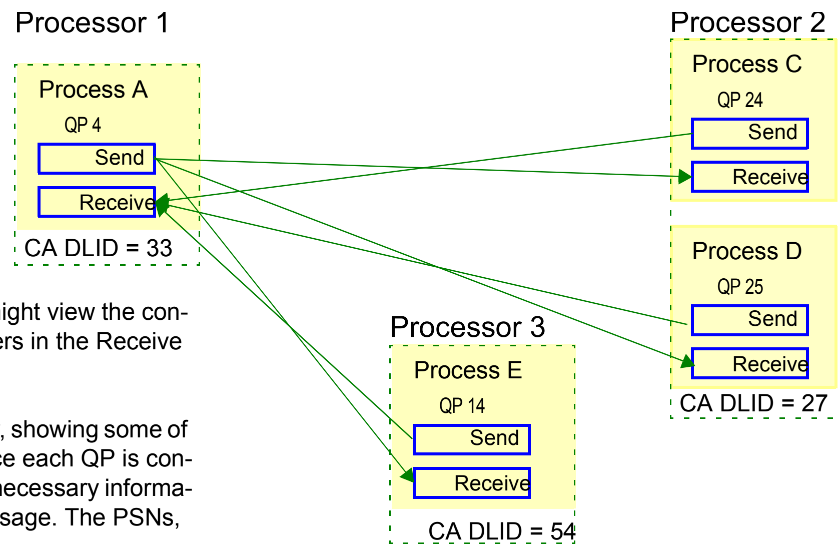
C9-201: Devices that source and sink UD messages shall meet the requirements of the basic Unreliable Services (see 9.8 Unreliable Service on page 341 through 9.8.1 Validating and Executing Requests on page 341).

C9-202: Devices that source and sink IBA UD messages shall meet the requirements specified in 9.9 Error detection and handling on page 362.

This figure shows two views of Unreliable Datagram QPs. Process A on Processor 1 communicates with three processes: processes C and D on Processor 2 and process E on processor 3.

The view on the right shows how software might view the connection. Buffers in the Send Q flow into buffers in the Receive Queue on the connected QP.

The lower view gives a hardware centric view, showing some of the state maintained per connected QP. Since each QP is connected to nothing, the destination and other necessary information (SL, DGID etc.) is picked with each message. The PSNs, while generated, are not checked.



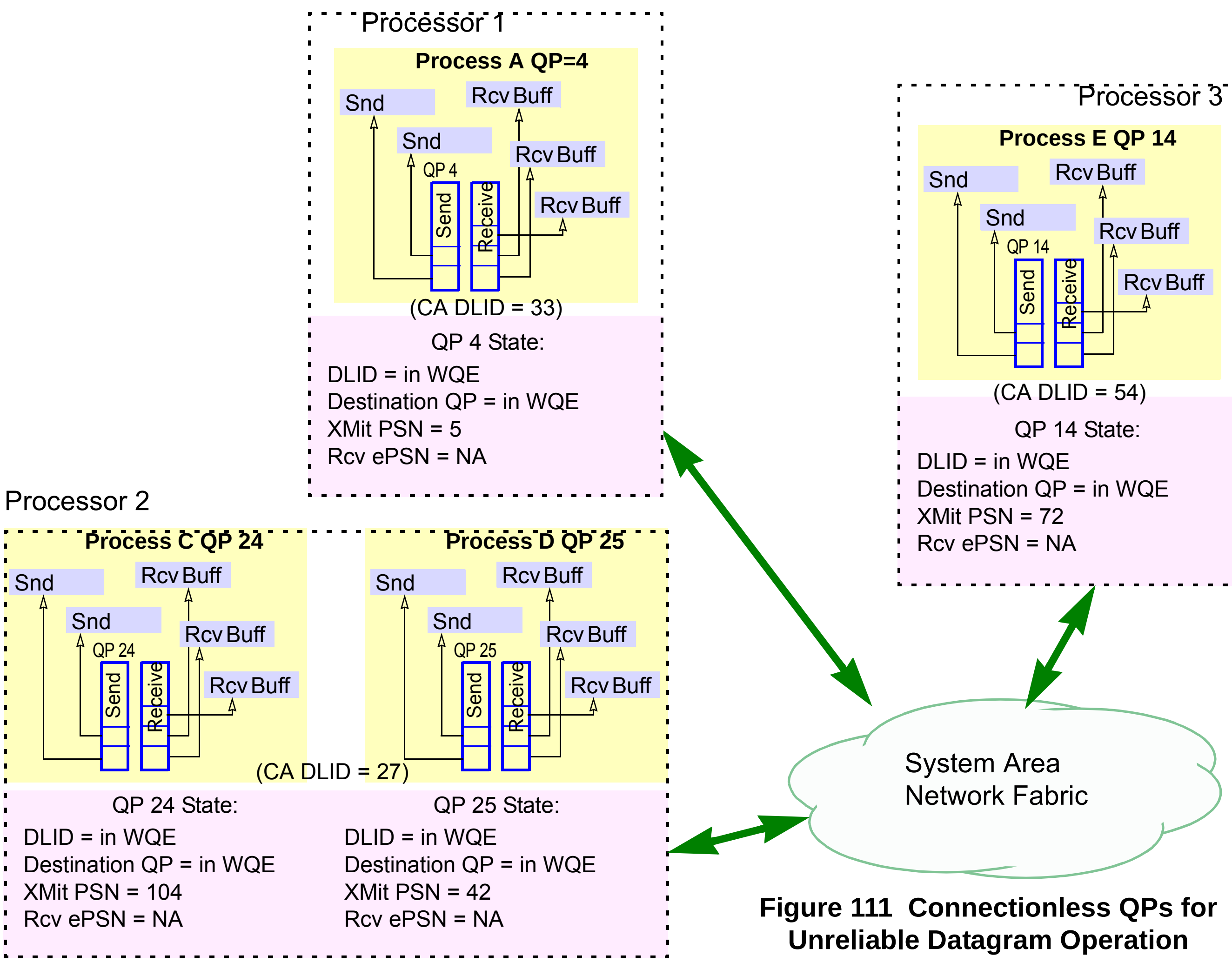


Figure 111 Connectionless QPs for Unreliable Datagram Operation

9.8.3.1 REQUESTER BEHAVIOR

This section specifies the requester's required behavior when generating request packets.

C9-203: Devices that source UD messages shall meet the requirements specified in 9.8.3.1.1 Generating PSN on page 358 and 9.8.3.1.2 Completing a Message Send on page 358 while sending UD messages.

9.8.3.1.1 GENERATING PSN

C9-204: For each request message on a UD transport service, the requester shall generate PSNs that is an increment of "1" (modulo 2^{24}) of the PSN value of the preceding request packet. The incrementing of PSN is not required for QP0 and QP1.

The initial PSN value shall be loaded by the transport's client while the send queue is in the Initialized state and may be initialized to any 24-bit value. While in the process of transmitting request packets, the transport layer shall modify (update) the PSN only when the send queue is in the Ready to Send state.

9.8.3.1.2 COMPLETING A MESSAGE SEND

The requester shall consider a message Send complete when it has:

- Committed the last byte of the VCRC field of the packet to the wire, and detected no local errors associated with the message transfer.
- Detected a local error associated with the message transfer that causes the requester to terminate sending the request.

Note that at the time that the requester completes the send WQE, the state of the memory at the responder is unknown. Likewise, if the requester detects a local error while sending the request packet, the state of the responder's memory is unknown.

9.8.3.2 RESPONDER BEHAVIOR

This section specifies the responder's required behavior when receiving inbound requests.

9.8.3.2.1 RESPONDER - VALIDATING THE PSN

o9-144: For UD transport service, the responder may ignore the PSN field.

Some applications (e.g. multicast-based media streaming) may derive benefit from having the responder validate the PSN sequence to detect out-of-sequence packets. It is permissible for a responder implementation to do so, but is outside the scope of the IBA specification.

9.8.3.2.2 RESPONDER - LENGTH VALIDATION

C9-205: Before executing the request, the responder shall validate the Packet Length field of the LRH and the PadCnt of the BTH as described in 9.8.3.2.2: Responder - Length Validation.

The following characteristics shall be validated:

- The Length fields shall be checked to confirm that there is sufficient space available in the receive buffer specified by the receive WQE.
- The packet payload length must be between zero and PMTU bytes inclusive in size.

If a packet is detected with an invalid length, the request shall be an invalid request and it shall be silently dropped by the responder as specified in Section 9.9.3 Responder Side Behavior on page 375. The responder then waits for a new request packet.

9.8.3.2.3 RESPONDER OPCODE VALIDATION

C9-206: For UD, the responder shall validate the BTH:OpCode for the requested function (SEND) is supported by this receive queue and is not reserved before executing the request else the request is invalid.

C9-207: If a UD receive queue does not have an entry to hold an inbound SEND request, the request is invalid.

If the request is invalid, it shall be silently dropped by the responder as specified in Section 9.9.3 Responder Side Behavior on page 375.

9.8.3.2.4 RESPONDER - LOCAL OPERATION VALIDATION

A valid inbound request may still fail to complete due to a failure that is local to the responder, e.g. local memory translation error while accessing local memory. A local error may cause the receive queue to transition to the error state. See 9.9.3 Responder Side Behavior on page 375 for additional details.

9.8.3.2.5 COMPLETING A MESSAGE RECEIVE

The responder considers a given inbound message completed successfully when it has:

- Committed the message payload to the local fault zone without error
- Successfully completed all appropriate validity checks (including variant and invariant CRC).

A failure detected during any of these steps may or may not cause the associated WQE to be completed in error. In some cases, such as an opcode or length error, no WQE will be consumed by the responder. Note that, in the presence of errors, it is not possible to guarantee the state of the responder's memory. Some or all of a given packet may have been committed to the responder's memory before the error is detected. Once an inbound message receive is completed successfully, the responder completes the current WQE.

9.8.4 RAW DATAGRAMS

The previous several sections describe the different transport protocols defined by the IBA specification. In addition to these, IBA allows other protocols to be carried by an IBA subnet. IBA datagrams that encapsulate such traffic are referred to as Raw Datagrams.

IBA defines two different methods to support Raw Datagrams. In Section 7.7.5 Link Next Header (LNH) - 2 bits on page 167 two bits in the local route header are used to specify the next header after the LRH. The following table describes the two LNH encodings that describe Raw Datagrams.

Link Next Header LNH(1:0)		Structure of the Raw Datagram			
IBA_Transport	GRH (IPv6) header				
0	1	LRH	IPv6	Packet Payload	VCRC
0	0	LRH	RWH	Packet Payload	VCRC

Figure 112 Raw Datagrams

The first method of encoding a Raw Datagram is used only for IPv6 datagrams. The packet payload may contain any transport or network protocol defined by the IETF's encoding of the IPv6 header's "next header" field excluding any encoding indicating the next header is an IBA transport header.

C9-208: CAs shall not generate an outbound packet and will discard any inbound packet whose LRH indicates a Raw Datagram and whose IPv6 "next header" indicates an IBA transport. TCAs may report this error in any manner they choose.

The second method of encoding a Raw Datagram uses the IBA defined raw header (RWH). The RWH contains the 16-bit Ethertype field - the RWH is described in section 5.3 Raw Packet Format on page 135. The RWH is used to define the protocol header encapsulated in the packet payload. In general, the second method is used to allow protocols not supported by the IPv6 next header - it should be noted that either method may be used to transport IPv6 datagrams.

o9-145: If a CA implements Raw Datagram support, the Packet Payload of Raw Datagrams must always be of a modulo 4 size, since the LRH Packet length describes the length in 4 byte increments. Should the encapsulated payload size not be a multiple of 4 bytes, the payload shall be padded to a multiple of 4 bytes.

o9-146: If a CA implements Raw Datagram support, the QPs used to inject and consume Raw Datagrams shall be locally managed, i.e. the association of a QP with a given Raw Datagram service is implementation dependent.

o9-147: If a CA implements Raw Datagram support, it may support one or more QPs for Raw Datagram operations.

9.8.4.1 RAW DATAGRAM PACKET SIZE

The IBA MTU defines the maximum size of an IBA transport's data payload. The maximum size of an IBA packet is MTU+124 bytes[1] (see 7.7.8 Packet Length (PktLen) - 11 bits on page 167).

o9-148: If a CA implements Raw Datagram support, and since a Raw datagram does not use IBA transport headers, raw datagrams may have a packet payload larger than the supported MTU (see Figure 112 Raw Datagrams on page 360). The table below summarizes the maximum packet payload (and the corresponding value for the LRH PktLen field) for each of the two raw datagram types.

Table 54 Maximum Raw Datagram Packet Payload

MTU	IPv6 Raw Datagram		RWH Raw Datagram	
	Largest Possible Packet Payload[a]	Corresponding PktLen Value	Largest Possible Packet Payload[b]	Corresponding PktLen Value
256	332 Bytes	95	368 Bytes	95
512	588 Bytes	159	624 Bytes	159
1024	1100 Bytes	287	1136 Bytes	287
2048	2124 Bytes	543	2160 Bytes	543
4096	4172 Bytes	1055	4208 Bytes	1055

a. largest possible IPv6 raw packet payload = MTU + 124 (the largest packet header/CRC size) - 8 (LRH size) - 40 (IPv6 header size)
b. largest possible RWH raw packet payload = MTU + 124 (the largest packet header/CRC size) - 8 (LRH size) - 4 (RWH header size)

1. The 124B maximum packet header/CRC byte count does not include the VCRC field.

9.9 ERROR DETECTION AND HANDLING

IBA uses a layered error management architecture (LEMA) approach. Each level is responsible for detecting and managing errors appropriate to that layer before passing the packet or message up to the next layer in the stack.

Thus the transport layer responds to errors particular to the transport including errors in the packet header and failures to correctly transport a message.

Errors detected in the transport layer are reported to the transport's client. In this section, the interface between the transport layer and its client is shown conceptually as the send or Receive Queue. In the case of an HCA, the transport indicates errors to its client by writing a completion code to a Completion Queue Entry (CQE) on the Completion Queue (CQ). As usual TCAs are free to report errors (or not) as they see fit.

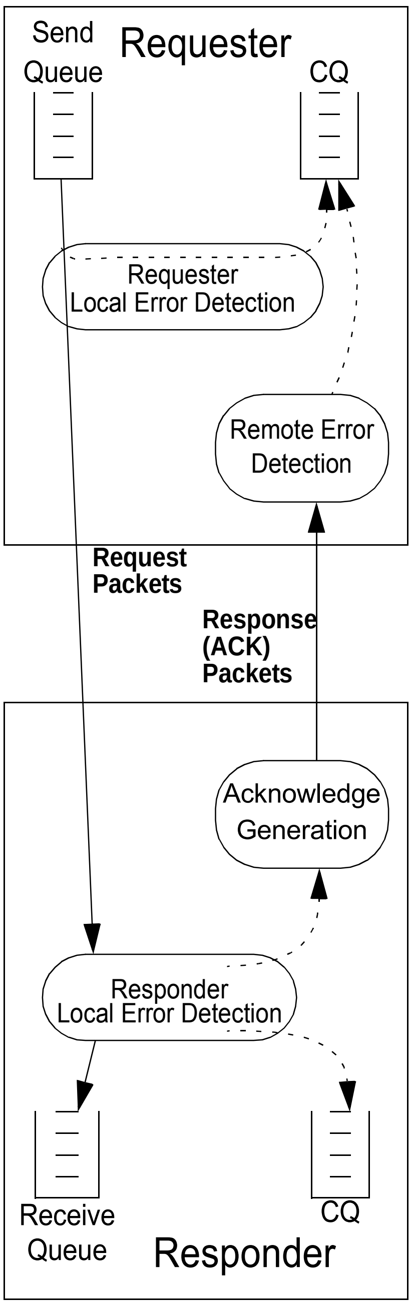


Figure 113 Requester / Responder Error Detection

In order to simplify the discussion, error behavior is discussed separately for the requester and responder ends. This causes a slight amount of duplication between the summary tables in the following sections describing the errors for the requester and responder side. Specifically, overlaps occur when an error is detected by the responder and reported to the requester. These areas of overlap, however, are strictly confined to reliable classes of service.

Errors that are reported by the requester to its client fall into one of two classes. The first are Locally Detected errors; i.e., errors that are detected solely by the requester side. An example of a locally detected error is a protection fault detected by the requester while accessing its own local memory during a send request.

The second class is remotely detected errors, which are those errors detected by the responder and reported to the requester via a NAK syndrome in an Response packet. Remotely detected errors only apply to the reliable classes of service (reliable connected and reliable datagram).

Whereas there were two classes of errors for the requester side (locally and remotely detected), there are only locally detected errors on the responder side.

In response to a locally detected error, the responder side may be required to report the error to the requester, or it may be required to report the error to its local client, or both, or neither. The choice of to whom the error is reported is governed by the class of service (reliable versus unreliable), and the specific error that is detected.

The key focus of the following sections is to categorize all errors according to how errors are reported to the transport layer's client, and the behavior that the send (receive) queue must exhibit following detection of an error. Thus, this section is categorized according to not only where an error is detected, but to whom it is reported.

9.9.1 Reporting Errors to the Verbs Layer

For an HCA, the IBA software interface defines three types of errors that can be reported through the verbs layer. These are called immediate errors, completion errors, and asynchronous errors. Of those three types, the transport layer is only capable of reporting completion errors or asynchronous errors. This is because immediate errors are detected by the verbs layer before the WQE ever gets posted to the transport layer. Table 55 summarizes the types of errors that an IBA transport can detect and report to the verbs layer. For more information on these error types, see 10.10.2 Error Handling Mechanisms on page 461.

Table 55 Software Error Types Detected by Transport Layer

IBA Software Defined Error Types	Detected by IBA Transport
Immediate Errors	no
Completion Errors - Interface check	yes
Completion Errors - Processing error	yes
Asynchronous Errors - Affiliated type	yes
Asynchronous Errors - Unaffiliated type	yes

There are two classes of completion errors: Interface checks and processing errors. An interface check is an error in the information supplied to the Channel Interface detected before data is placed onto the link. A processing error is an error encountered during the processing of the work request by the Channel Interface.

9.9.2 REQUESTER SIDE ERROR BEHAVIOR

As indicated above, the requester detects errors originating locally or remotely.

9.9.2.1 REQUESTER SIDE ERROR DETECTION - LOCALLY DETECTED ERRORS

A locally detected error reflects either an error condition that has occurred in the requester's channel interface, a missing response from the responder side (timeout) or excessive retries for sequence errors or RNR NAKs.

Locally detected errors at the requester can occur during request packet generation, during the processing of response packets, or due to a timeout.

C9-209: For an HCA requester using RC, UC, or UD service, and for a TCA requester using UD service, the requester shall behave as follows. For locally detected transport errors that are detected during transmission of request packets, a CA shall stop transmission for the affected QP, shall store the state associated with the error until any previous incomplete WQEs are completed, and finally complete the affected WQE. The QP shall be put into the error state if the error type requires this.

o9-149: If a TCA requester implements Reliable Connection or Unreliable Connection service, it shall behave as follows. For locally detected transport errors that are detected during transmission of request packets, a CA shall stop transmission for the affected QP, shall store the state associated with the error until any previous incomplete WQEs are completed, and finally complete the affected WQE. The QP shall be put into the error state if the error type requires this.

o9-150: If a CA requester implements Reliable Datagram service, it shall behave as follows. For locally detected transport errors that are detected during transmission of request packets, a CA shall stop transmission for the affected EEC, shall store the state associated with the error until any previous incomplete WQEs are completed, and finally complete the affected WQE. The EEC shall be put into the error state if the error type requires this.

This is required to maintain the ordered completion of WQEs and to ensure that the error is properly reported in the WQE where the error occurred.

9.9.2.1.1 REQUESTER ERROR RETRY COUNTERS

C9-210: For an HCA using Reliable Connection service, in order to detect excessive retries, the requester shall maintain the RNR NAK and Error retry counters that perform the logical functions described in 9.9.2.1.1 Requester Error Retry Counters on page 364.

o9-151: If a CA requester implements Reliable Datagram service, or if a TCA requester implements Reliable Connection service, the requester shall behave as follows. In order to detect excessive retries, the requester shall maintain the RNR NAK and Error retry counters that perform the logical functions described in 9.9.2.1.1 Requester Error Retry Counters on page 364.

Implementations may implement these retry counters in any way they choose, but for clarity, they are here described as down counters, initialized to the number of retries allowed before terminating the operation and creating the final completion error. See 10.2.3.3 Modifying Queue Pair Attributes on page 401 for the programming of these counters in an HCA.

The RNR NAK retry counter is decremented each time the responder returns an RNR NAK. If the requester's RNR NAK retry counter decrements to zero, an RNR NAK retry error occurs. Each time an RNR NAK is cleared (i.e., an acknowledge message other than an RNR NAK is returned), the retry counter is reloaded. An exception to the following is if the RNR NAK retry counter is set to 7. This value indicates infinite retry and the counter is not decremented.

The Error retry counter is decremented each time the requester must retry a packet due to a Local Ack Timeout, NAK-Sequence Error, or Implied NAK. If the requester's retry counter decrements to zero, one of two things may be implemented.

If Automatic Path migration is not supported, or has already been completed, a "Transport Retry Counter Exceeded" error shall be reported in the completion.

o9-152: If a CA supports Automatic Path Migration, then, following a potentially recoverable error and its retries, the requester may migrate the connection or EE context and perform the Error retries again before finally reporting the completion in error. See 17.2.8 Automatic Path Migration on page 836 for more information.

Each time a packet is properly acknowledged, the retry counter shall be reloaded.

9.9.2.2 Requester Side Error Detection - Remotely Detected Errors

A remotely detected error occurs when the responder reports an error to the requester. Remotely detected errors are unique to reliable classes of service.

Remotely detected errors are reported via a NAK code carried in an acknowledge message. However, not all NAK codes result in an error being reported to the requester's client.

Of the possible NAK codes, two (NAK-Sequence Error and NAK-RNR) indicate operations that should be retried automatically by the requester.

The NAK codes other than NAK-Sequence Error and NAK-RNR indicate failures that must be reported to the requester's client immediately and cannot be retried.

9.9.2.3 SUMMARY - REQUESTER SIDE ERROR BEHAVIOR

Table 56 lists all errors that are detected by the requester, including both locally and remotely detected errors. If the error is detected locally by the requester, the column labelled "Syndrome" contains the notation "locally detected error". If the error is detected remotely by the responder, this column lists the NAK syndrome that was returned by the responder. The fault behavior class specifies the actions that the requester takes to report the error to its client.

Each fault behavior class is specified below in Sections 9.9.2.4.1 through 9.9.2.4.5. For convenience, the six possible classes of fault behaviors are summarized in Table 57 Summary of Requester Fault Behavior Classes on page 369 below.

C9-211: For the implemented subset of transport services, requesters shall conform to the error behavior as specified in Table 56. Also, the requester's send queue shall behave as specified for each Fault Behavior Class shown in Table 57 and each of the Requester Class Fault descriptions.

Table 56 Requester Side Error Behavior

Error	Description	Syndrome	Requestor Fault Behavior Class
Packet sequence error. Retry limit not exceeded.	Responder detected a PSN larger than it expected. Requester may retry the request.	NAK-Sequence Error	RC: Class A RD: Class A else: NA
Packet sequence error. Retry limit exceeded.	Responder detected a PSN larger than it expected. The requestor performed retries, and automatic path migration and additional retries, if applicable, but all attempts failed.	NAK-Sequence Error	RC: Class B RD: Class D else: NA
Implied NAK sequence error. Retry limit not exceeded.	Requestor detected an ACK with a PSN larger than the expected PSN for an RDMA READ or ATOMIC response. Requester may retry the request.	locally detected error	RC: Class A RD: Class A else: NA
Implied NAK sequence error. Retry limit exceeded.	Requestor detected an ACK with a PSN larger than the expected PSN for an RDMA READ or atomic response. The requestor performed retries, and automatic path migration and additional retries, if applicable, but all attempts failed.	locally detected error	RC: Class B RD: Class D else: NA
Local Ack Timeout error. Retry limit not exceeded.	No ACK response from responder within timer interval. Requester may retry the request.	locally detected error	RC: Class A RD: Class A else: NA
Local Ack Timeout error. Retry limit exceeded.	No ACK response within timer interval. The requestor performed retries, and automatic path migration and additional retries, but all attempts failed.	locally detected error	RC: Class B RD: Class D else: NA
RNR NAK Retry error. Retry limit not exceeded.	Responder returned RNR NAK. Requestor may retry the request.	RNR-NAK	RC: Class A RD: Class A else: NA
RNR NAK Retry error. Retry limit exceeded.	Excessive RNR NAKs returned by the responder. Requestor retried the request "n" times, but received RNR NAK each time.	locally detected error	RC: Class B RD: Class B else: NA
Unsupported OpCode.	Responder detected an unsupported OpCode.	NAK-Invalid Request	RC: Class B RD: Class B else: NA
Unexpected OpCode.	Responder detected an error in the sequence of OpCodes, such as a missing "Last" packet. Note: there is no PSN error, thus this does not indicate a dropped packet.	NAK-Invalid Request	RC: Class B RD: Class B else: NA
Local Memory Protection Error.	Requester detected an implementation specific memory protection error in its local memory subsystem.	locally detected error	All: Class B

Table 56 Requester Side Error Behavior

Error	Description	Syndrome	Requestor Fault Behavior Class
R_Key Violation	Responder detected an invalid R_Key while executing an RDMA Request	NAK-Remote Access Error	RC: Class B RD: Class B else: NA
Remote Operation Error	Responder encountered an error, (local to the responder), which prevented it from completing the request.	NAK-Remote Operation Error	RC: Class B RD: Class B else NA
Local Operation Error[a] - affiliated	An error occurred in the requester's local channel interface that can be associated with a certain WQ or EEC.	locally detected error	All: Class B
Local Operation Error[a] - unaffiliated	An error occurred in the requester's local channel interface that cannot be associated with a certain WQ or EEC.	locally detected error	All: Class C
Local RDD Violation	Requester's EE Context detected an invalid RDD on an outbound packet	locally detected error	RD: Class B else NA
Remote RDD Violation	Responder's Receive Queue detected a RDD violation	NAK-Invalid RD Request	RD: Class B else NA
Remote Q_Key Violation	Responder's Receive Queue detected a Q_Key violation	NAK-Invalid RD Request	RD: Class B else NA
Length error	RDMA READ response message contained too much or too little payload data.	locally detected error	RC: Class B RD: Class B else NA
Bad response	Unexpected opcode for the response packet received at the expected response PSN.[b]	locally detected error	RC: Class B RD: Class B else NA
Ghost Acknowledge	Requester received an acknowledge message at other than the expected response PSN.	locally detected error	RC: Class E RD: Class E Else NA
CQ overflow	Despite actual execution of the message, and acknowledgement, the completion notification could not be written to the CQ.	locally detected error	All: Class F

a. Local operations errors tend to be very implementation specific; not all CA's may have or detect these.
b. For example; RDMA read instead of Acknowledge, NAK code in AETH of an RDMA read, or "RDMA READ Response last" instead of middle.

Table 57 Summary of Requester Fault Behavior Classes

Fault Behavior Class	Current Send Queue WQE	Subsequent Send Queue WQEs	Final Send Queue State
Requester Class A	no impact	no impact	no change
Requester Class B	completed in error	flushed	error state
Requester Class C	no impact - unaffiliated	no impact	no change[a]
Requester Class D[b]	completed in error	flushed	error state
Requester Class E[c]	no impact	no impact	no change
Requester Class F	unknown	unknown	error state

a. It is possible that this class of error will render the entire HCA unable to continue work.
b. Classes B and D are similar, however Class D applies to reliable datagram service only and also specifies that the requester's EE Context transition to the error state.
c. Classes A and E look similar, but Class A requires a retry, Class E results in no action.

9.9.2.4 REQUESTER SIDE ERROR RESPONSE

There are five different sets of error response behaviors that the requester must implement. Which behavior is executed for any given error is shown above in Table 56. This section specifies the error response behaviors.

9.9.2.4.1 REQUESTER CLASS A FAULT BEHAVIOR

Class A errors are those that are recoverable by the transport through a retry mechanism. If the retry succeeds, there is no visible impact to the transport's client (e.g. verbs layer).

The only Class A errors are Packet Sequence Error, Implied NAK sequence error, Local Ack Timeout error and an RNR NAK. Packet Sequence Error and RNR NAK are both remotely detected. A Local Ack Timeout error and an Implied NAK sequence error are detected locally by the requester.

C9-212: For an HCA using Reliable Connection service, each time the transport retries a Requester Class A error, it shall decrement a retry counter. There is one retry counter associated with Packet Sequence Errors and Local Ack Timeout errors, and a different retry counter associated with RNR NAKs. As long as the retry count has not expired the transport may continue to retry these errors. The protocol for retrying these errors is given in Section 9.7 Reliable Service on page 247.

o9-153: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, each time the requester retries a Requester Class A error, it shall decrement a retry counter. There is one retry counter associated with Packet Sequence Errors and Local Ack Timeout errors, and a different retry counter associated with RNR NAKs. As long as the retry count has not expired the transport may continue to retry these errors. The protocol for retrying these errors is given in Section 9.7 Reliable Service on page 247.

C9-213: For an HCA requester using Reliable Connection service, since Requester Class A errors are recoverable, the requester shall not report them to the transport's client unless the retry count expires.

o9-154: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, since Requester Class A errors are recoverable, the requester shall not report them to the transport's client unless the retry count expires.

See Section 10.10.2.2 Completion Errors on page 461 for a discussion of how errors are reported for an HCA once the retry count has expired.

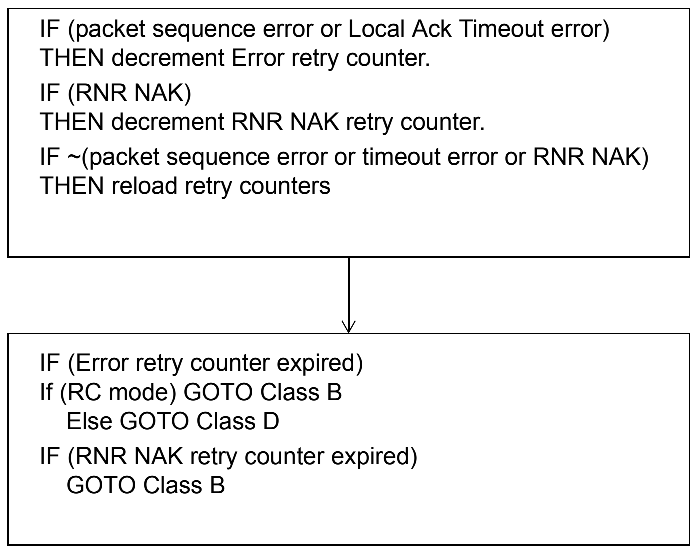


Figure 114 Requester Class A Fault Behavior

9.9.2.4.2 REQUESTER CLASS B FAULT BEHAVIOR

C9-214: In response to a Requester Class B error, for services other than Reliable Datagram, the requester shall complete the current WQE in error, transition the Send Queue to the error state and mark any subsequent WQEs posted to the Send Queue as flushed.

o9-154.a1: For CAs that implement Reliable Datagram service, the requestor, in response to a Requester Class B error, shall perform the actions described in Section 9.7.8.5 Handling QP errors - RESYNC on page 333. If, following the RESYNC process described, the message is still in error, the requester shall complete the current WQE in error, transition the Send Queue to the error state and mark any subsequent WQEs posted to the Send Queue as flushed.

For an HCA, the error is posted as “Completion - Processing type” with the appropriate error type (See 10.10.2.2 Completion Errors on page 461 for more details).

The queue shall be transitioned to the error state by the transport layer to prevent a race condition that can occur if the client (e.g. the verbs layer for an HCA) posts further WQEs to the Send Queue before it discovers that an error has occurred. This is consistent with the Send Queue state diagram as shown in Figure 122 QP/EE Context State Diagram on page 412.

Finally, all WQEs in the Send Queue behind the failed WQE are also completed with the “Completed - Flushed in Error” status.

For RC mode, note that some of these requests may have been committed to the wire by the requester, and may even have been executed and completed by the responder. It is not possible to prevent this since the responder may have executed the request before the requester detects a local error. Therefore, the responder’s local state must be considered unknown.

For reliable datagram service, the requester’s EE Context terminates the current message transfer, signals the error to the currently scheduled Send Queue, and removes the currently scheduled Send Queue from the scheduler. The EE Context then schedules the next Send Queue requesting service and proceeds. The Send Queue which caused the error behaves as described above.

C9-215: While the Send Queue is in the error state, it must silently discard any acknowledge messages that arrive.

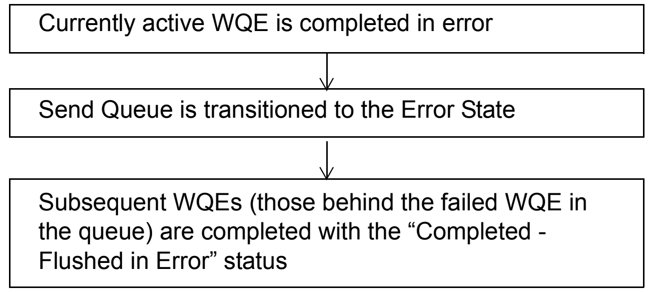


Figure 115 Requester Class B Fault Behavior

9.9.2.4.3 REQUESTER CLASS C FAULT BEHAVIOR

Since a Class C error cannot be associated with any particular WQE, it is not possible to mark a specific WQE as completed in error. For an HCA, the error posted is, "Asynchronous - Affiliated Error".

C9-216: If the Requester Class C error can be associated with a QP, the Send Queue shall be transitioned to the error state, and all uncompleted WQEs are completed with the "Completed - Flushed in Error" status.

o9-155: If a CA requester implements Reliable Datagram service, and if the Requester Class C error can be associated with an EE Context, its send side shall be transitioned to the error state, and for an HCA, the error posted is, "Asynchronous - Affiliated Error". See 10.10.2.2 Completion Errors on page 461 for more details on HCA error reporting.

9.9.2.4.4 Requester Class D Fault Behavior

A Class D error only occurs for reliable datagram service.

o9-156: If a CA requester implements Reliable Datagram service, it shall behave as follows. For the Requester Class D error class, the transport shall transition the requester's EE Context to the error state, terminate the current message transfer, signal the error to the currently scheduled Send Queue, and de-queue the currently scheduled Send Queue. While remaining in error state, the EE Context continues to transition to error state any other Send Queue requesting service.

Each Send Queue (QP) behaves as though for a Class B error; it marks its current WQE as completed in error, transitions the QP to the error state, and flushes all subsequent WQEs.

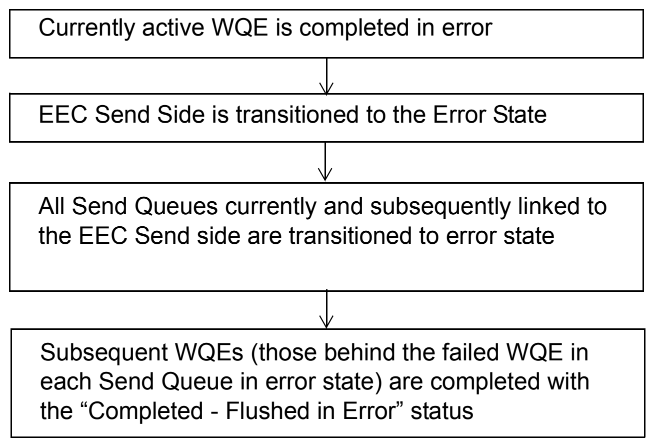


Figure 116 Requester Class D Fault Behavior

9.9.2.4.5 REQUESTER CLASS E FAULT BEHAVIOR

A Class E error occurs when the requester receives an acknowledge message with a PSN which does not match its expected PSN. These errors occur only for reliable classes of service.

These errors are not reported to upper layers.

An acknowledge message with an unexpected PSN is presumed to represent a “ghost” acknowledge message, or a duplicate acknowledge message.

A ghost acknowledge message is an acknowledge message that has been in the fabric long enough that it has survived the destruction of a connection and the subsequent establishment of a new connection.

A duplicate acknowledge message occurs when the requester, believing that its original request message is lost in the fabric, re-sends the request message. If both request messages eventually arrive at the responder, the responder may generate an acknowledge message for each of them.

C9-217: For an HCA requester using Reliable Connection service, in response to a Requester Class E error, the requester shall drop the acknowledge message. There is, however, an exception to this rule. For reliable connected service, a duplicate acknowledge message may be used by the responder to carry end-to-end flow control credits to the requester (an “unsolicited acknowledge”). Thus, if the PSN of the acknowledge message is one less than the requester’s expected PSN, the requester must recover the end-to-end credits and discard the remainder of the message. This behavior is detailed in section 9.7.7.2 End-to-End (Message Level) Flow Control on page 314.

o9-157: If a TCA requester implements Reliable Connection service, or if a CA requester implements Reliable Datagram service, in response to a Requester Class E error, the requester shall drop the acknowledge message.

It should be noted that even if the Acknowledgment was an actual ghost, with wrong credits, the credit mechanism would eventually recover with no errors reported to the upper layers.

9.9.2.4.6 REQUESTER CLASS F FAULT BEHAVIOR

C9-218: A Requester Class F error occurs when the CQ is inaccessible or full and an attempt is made to complete a WQE. The Affected QP shall be moved to the error state and an affiliated asynchronous error generated. The current WQE and any subsequent WQEs are left in an unknown state. See 10.10.2.3 Asynchronous Errors on page 462.

9.9.3 Responder Side Behavior[1]

Table 58 lists the errors that must be detected by the responder, and the Fault Behavior Class for each error. The Fault Behavior Class specifies whether the responder returns a NAK code, whether the error is reported to the local client, and the subsequent behavior of the Receive Queue. The syndrome column lists the NAK code that is returned to the requester.

For convenience, a summary of the fault behavior classes is shown in Table 59 Summary of Responder Fault Class Behaviors on page 377.

The error detection for reliable service is described in section 9.7 Reliable Service on page 247, and error detection for unreliable service is specified in section 9.8 Unreliable Service on page 341.

C9-219: For the implemented subset of transport services, responders shall conform to the error behavior as specified in Table 58. Also, the responder's Receive queue shall behave as specified for each Fault Behavior Class shown in Table 59 and each of the sub-sections below.

Table 58 Responder Error Behavior Summary

Error	Description	Service	Syndrome	Fault Behavior Class
Malformed WQE	Responder detected a malformed Receive Queue WQE while processing the packet.	RC, RD	NAK-Remote Operational Error	Responder Class A
		Else	NA	Responder Class A
Unsupported or Reserved OpCode	Inbound request OpCode was either reserved, or was for a function not supported by this QP. E.G. RDMA or ATOMIC on QP not set up for this. For RC this is "QP Async affiliated"	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class B
		else	NA	Responder Class D
Misaligned ATOMIC	VA does not point to an aligned address on an atomic operation	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class B
Too many RDMA READ or ATOMIC Requests	There were more requests received and not ACKed than allowed for the connection	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class B
Out of Sequence Request Packet	PSN of the inbound request is outside the responder's valid PSN window.	RC, RD	NAK-Sequence error	Responder Class B
		UC	NA	Responder Class D
Out of Sequence OpCode, current packet is "first" or "Only"	The Responder detected an error in the sequence of OpCodes; a missing "Last" packet	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class B
		UC	NA	Responder Class D1
Out of Sequence OpCode, current packet is not "first" or "Only"	The Responder detected an error in the sequence of OpCodes; a missing "First" packet	RC	NAK-Invalid Request	Responder Class C
		RD		Responder Class B
		UC	NA	Responder Class D

1. For Unreliable services, a better title might be Receiver side Behavior.

Table 58 Responder Error Behavior Summary

Error	Description	Service	Syndrome	Fault Behavior Class
RESYNC Opcode incomplete WQE	The Responder has a partially complete WQE when a valid RESYNC arrives "Requestor Aborted"		NA	Responder Class E
R_Key Violation	Responder detected an R_Key violation while executing an RDMA request.	RC	NAK-Remote Access Violation	Responder Class C
		RD	NAK-Remote Access Violation	Responder Class B
		UC	NA	Responder Class D
Local QP Error	Responder detected a local QP related error while executing the request message. The local error prevented the responder from completing the request.	RC, RD	NAK-Remote Operational Error	Responder Class A
		Else	NA	Responder Class A
Q_Key Violation	Responder's Receive Queue detected an invalid Q_Key in the request message[a]	RD	NAK-Invalid RD Request	Responder Class B
		UD	NA	Responder Class D
Packet Header Violation	Responder detected a header violation that requires a silent drop as described in 9.6 Packet Transport Header Validation on page 236	RC, RD UC, UD	none	Responder Class D
RDD Violation	Responder's Receive Queue detected an invalid RDD	RD	NAK-Invalid RD Request	Responder Class B
Invalid Dest QP	Dest QP does not exist or is not configured for RD service	RD	NAK-Invalid RD Request	Responder Class B
Resources Not Ready Error	A WQE or other resource is not currently available.	RC,RD	RNR-NAK	Responder Class B
Length errors	1) Inbound "Send" request message exceeded the responder's available buffer space: "Local Length Error" 2) RDMA WRITE request message contained too much or too little payload data compared to the DMA length advertised in the first or only packet. 3) Payload length was not consistent with the opcode: a: 0 byte <= "only" <= PMTU bytes b: ("first" or "middle") == PMTU bytes c: 1byte <= "last" <= PMTU bytes	RC	NAK-Invalid Request	Responder Class C
		RD	NAK-Invalid Request	Responder Class F
		UC, UD (only 1, 3a)	NA	Responder Class D
Invalid duplicate ATOMIC Request	A duplicate ATOMIC request packet is received, but the PSN does not match the PSN of a saved ATOMIC Request.	RC, RD	none	Responder Class D
CQ overflow	Despite actual execution of the message, and acknowledgement, the completion notification could not be written to the CQ.	All	none	Responder Class G
Local EEC Error	Responder detected a local EEC related error while executing the request message. The local error prevented the responder from completing the request.	RD	none	Responder Class H

a. Q_Key violations require the incrementing of a counter and a potential trap as described in 10.2.4 Q_Keys on page 403

Table 59 Summary of Responder Fault Class Behaviors

Fault Behavior Class	NAK Codes Returned	Current Receive WQE[a]	Subsequent Receive WQEs	Final Receive Queue State
Responder Class A	For reliable services: Remote Operational Error	WQE completed in error	flushed	error state
Responder Class B	Invalid Request Remote Access Violation Sequence error RNR-NAK	no WQE consumed	no impact	no change
Responder Class C	Invalid Request	completed in error	flushed	error state
Responder Class D, D1	none	no WQE consumed	no impact	no change
Responder Class E	none	may be completed in error	no impact	no change
Responder Class F	Invalid Request	completed in error	no impact	no change
Responder Class G	none	unknown	unknown	error state
Responder Class H[b]	none	completed in error	no impact	no change

a. A WQE is only completed if open for Sends and RDMA WRITE with Immediate data.
b. This error class results in the EEC being put into the Error state.

9.9.3.1 RESPONDER SIDE ERROR RESPONSE

There are a total of eight classes of fault behavior described for the responder side. The fault behaviors are grouped according to whether or not an error is reported to the client on the responder side, whether or not the error is reported to the requester via a NAK code, and whether or not a WQE is consumed from the Receive Queue.

9.9.3.1.1 RESPONDER CLASS A FAULT BEHAVIOR

Class A errors are traceable to a poorly formed or invalid WQE, or other error associated with the receiver QP. These errors are not caused by the sender.

C9-220: For a Responder Class A error, the error shall be reported to the responder's client, the QP is placed into the error state, and, for reliable services, a "NAK-Remote Operational Error" is generated.

For Reliable Datagram service, the EEC continues operation.

If the responder is an HCA, these errors are reported to the verbs layer as a "Completion error - interface type", "Internal Consistency error". See 10.10.2 Error Handling Mechanisms on page 461 for a discussion of Completion errors.

If the responder detects a Class A error, its behavior is as follows:

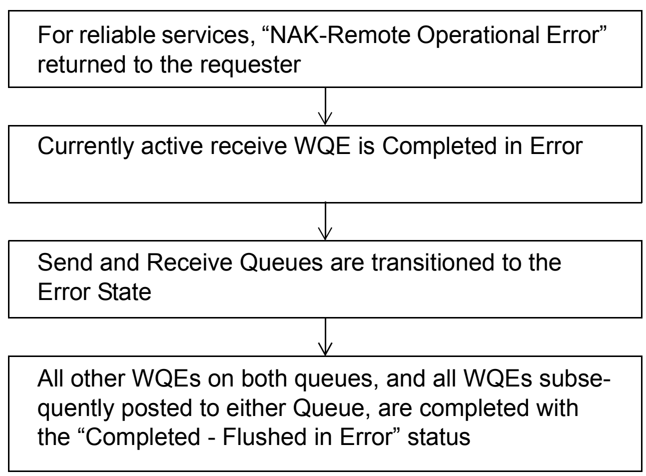


Figure 117 Transport Class A Responder Behavior

9.9.3.1.2 RESPONDER CLASS B FAULT BEHAVIOR

Class B errors are reported to the requester, but are not reported to the responder's local client.

C9-221: For an HCA requester using Reliable Connection service, and for a Responder Class B responder side error, the transport shall generate a NAK code, but shall not consume a WQE from the Receive Queue or transit the receive queue to the error state.

o9-158: If a TCA responder implements Reliable Connection service, or if a CA responder implements Reliable Datagram service, it shall behave as follows. For a Responder Class B responder side error, the transport shall generate a NAK code, but shall not consume a WQE from the Receive Queue or transit the receive queue to the error state.

Note that this fault behavior class is limited to reliable services only.

If the responder detects a Class B error, it behaves as follows:

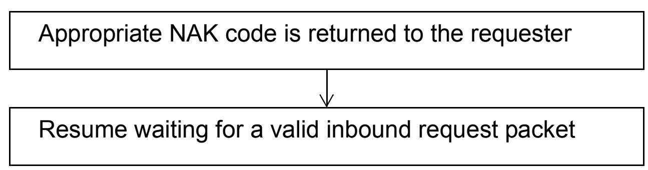


Figure 118 Responder Class B Fault Behavior

9.9.3.1.3 RESPONDER CLASS C FAULT BEHAVIOR

C9-222: For an HCA responder using Reliable Connection service, for a Class C responder side error, the error shall be reported to the responder's client and the QP is placed into the error state. A Class C error shall also be reported to the requester by generating a NAK-Invalid Request. In the case of an HCA, the error reported in a Receive queue completion is a "Completion - Process Error". If the current operation does not use a receive WQE, then an affiliated asynchronous error is generated.

o9-159: If a TCA responder implements Reliable Connection service, for a Class C responder side error, the error shall be reported to the responder's client and the QP is placed into the error state. A Class C error shall also be reported to the requester by generating a NAK-Invalid Request.

The Receive Queue's behavior is as follows:

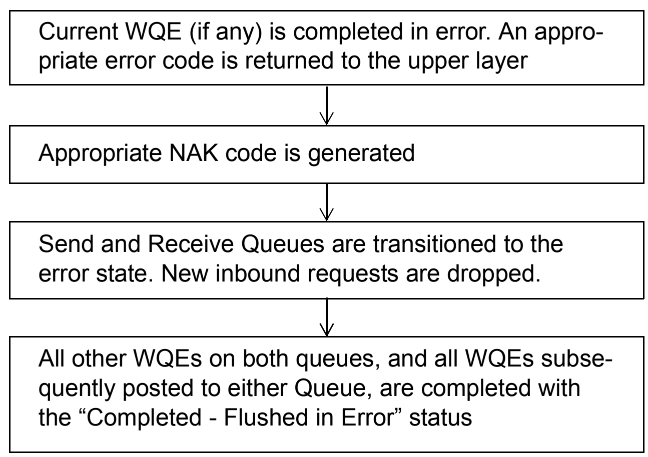


Figure 119 Transport Class C Receive Queue Behavior

See Section 10.10.2.2 Completion Errors on page 461 for more details on HCA error reporting.

9.9.3.1.4 RESPONDER CLASS D FAULT BEHAVIOR

C9-223: An inbound request packet which causes a Responder Class D error shall cause the Transport to respond as specified in 9.9.3.1.4: Responder Class D Fault Behavior.

In this case the transport shall:

- silently drop the packet
- Not generate an ACK or NAK code to the requester
- Not notify the responder's client
- terminate the current message without consuming the current receive WQE (if any)
- wait for the first packet of a new message (For reliable services, the new message must begin at the expected PSN.)

The Current WQE (if any) is reset to accept the next incoming Send or RDMA WRITE with Immediate message.

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, including the present packet (which caused the Class D error).

A "new message" is denoted by a packet with a BTH opcode of "first" or "only".

9.9.3.1.5 RESPONDER CLASS D1 FAULT BEHAVIOR

An inbound request packet which causes a Class D1 error only occurs in Unreliable Connection mode.

C9-224: For an HCA responder using Unreliable Connection service, an inbound request packet which causes a Responder Class D1 error shall cause the Transport to respond as specified in 9.9.3.1.5: Responder Class D1 Fault Behavior.

In this case the transport shall:

- silently drop the packet
- Not notify the responder's client
- terminate the current message without consuming the current receive WQE (if any)
- wait for the first packet of a new message (which may be greater than the expected PSN.)

If the present packet, (which caused the Class D1 error) has a BTH opcode of "first" or "only"; it shall be treated as the first packet of a new message.

The Current WQE (if any) shall be reset to accept the next incoming Send or RDMA WRITE with Immediate message.

"Current message" means all the packets received since the most recently received "first" or "only" OpCode, excluding the present packet (which caused the Class D1 error).

A "new message" is denoted by a packet with a BTH opcode of "first" or "only".

o9-160: If a TCA responder implements Unreliable Connection service, it shall conform to the Class D1 HCA responder behavior described in the preceding compliance statement.

9.9.3.1.6 RESPONDER CLASS E FAULT BEHAVIOR

This fault class is intended primarily for services where a failure of a particular request packet should not impact the ability of the Receive Queue to continue receiving messages.

o9-161: If a CA implements Reliable Datagram service, then a Responder Class E error shall cause the responder to do the following:

1) Abort the current message, if it is not complete. This is done by either:
   a) Reset the WQE so that it can be reused for a future message
   b) Mark the current WQE (if any) as completed in error “Requester Abort Error”
2) Receive the new inbound message. The Receive Queue shall continue operation without a transition to the error state.:

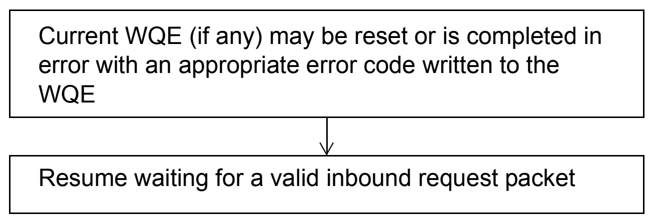


Figure 120 Transport Class E Receive Queue Behavior

9.9.3.1.7 RESPONDER CLASS F FAULT BEHAVIOR

A Class F error only occurs due to an invalid request length in Reliable Datagram service.

o9-162: If a CA implements Reliable Datagram service, then a Responder Class F error shall be reported both to the requester and (when a receive WQE is involved) to the responder's client. The Transport shall return the "NAK-Invalid request" to the requester, and complete the current WQE (if any) in error. The Receive Queue shall continue operation without a transition to the error state.

In the case of an HCA, the error reported is a "Completion - Process Error".

Both the EEC and destination QP remain in operation.

The Receive Queue's behavior is as follows:

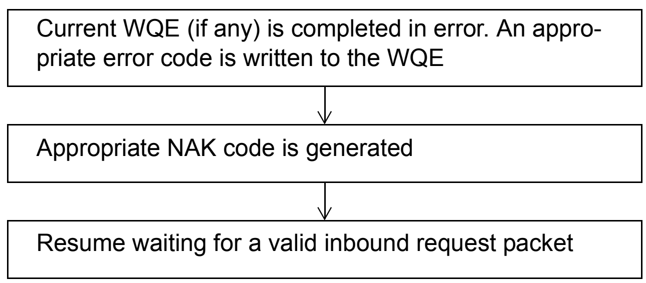


Figure 121 Transport Class F Receive Queue Behavior

9.9.3.1.8 RESPONDER CLASS G FAULT BEHAVIOR

A Class G error occurs when the CQ is inaccessible or full and an attempt is made to complete a WQE.

C9-225: A Responder Class G error occurs when the CQ is inaccessible or full and an attempt is made to complete a WQE. The Affected QP shall be moved to the error state and an affiliated asynchronous error generated. The current WQE and any subsequent WQEs are left in an unknown state.

See 10.10.2.3 Asynchronous Errors on page 462.

9.9.3.1.9 RESPONDER CLASS H FAULT BEHAVIOR

Class H errors are traceable to a local error associated with the receiver EEC. These errors are not caused by the sender.

For a Responder Class H error, the error shall be reported to the responder's client, and the EEC is placed in the error state.

The currently active WQE should be completed in error.

If the responder is an HCA, these errors are either reported to the verbs layer as a Completion error - Local Operation EE error, or as an Local Work Queue Catastrophic Affiliated Asynchronous Error pointing to the EEC. See 11.6 Result Types on page 538 for a discussion of Error codes.

9.10 Header and Data Field Source

9.10.1 Field source when generating packets

The following tables provide an indication of the source of the various header and data fields in the data packets for the various IBA services. The following terms are used in the table:

Link — This indicates the value is attached to the packet based on either a fixed value, or values dependent on the service, or values looked up based on parameters loaded into the logical port. Done by the link layer.

Tr — This indicates that the value is fixed or calculated by the transport layer.

QP — This indicates that the value is derived from the QP context

EE — This indicates that the value is derived from the EE context

QP+EE — This indicates that the values are derived from the QP and EE contexts

NA — Not Applicable

WQE — The value is directly or indirectly (via Address vector) derived from information in the WQE

Table 60 Packet Fields and Parameters by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
LRH VL	The VL to use for requests. Based on SL and the port SL to VL mapping table.	link	link	link	link	link	link
LRH LVer	The version of the link level. This field depends on the revision of the device.	link	link	link	link	link	link
LRH SL	The SL to use for requests	QP	QP	EE	WQE	WQE	WQE
LRH LNH IBA	IBA transport bit, indicates that BTH follows	1	1	1	1	0	0
LRH LNH GRH	GRH bit, indicates that a GRH follows	QP	QP	EE	WQE	1	0
LRH DLID	Destination local ID used for routing	QP	QP	EE	WQE	WQE	WQE
LRH Packet Length	Length of the local packet; calculated by the transport based on the message length.	WQE	WQE	WQE	WQE	WQE	WQE
LRH SLID (high bits not covered by LMC)	Source local ID in outgoing packets. From the port. With LMC low order bits (0s) added, the value is called "Base LID".	link	link	link	link	link	link
LRH SLID (low bits covered by the LMC)	Source logical ID in outgoing packets. These LMC (as a number) bits are called the "path" bits.	QP	QP	EE	WQE	WQE	WQE
GRH IPVer"	CA's set to 6	Tr	Tr	Tr	Tr	Tr	NA
GRH Tclass	CA's set to 0; it will then be loaded with another value at the first encountered router. Alternately set according to application.	QP	QP	EE	WQE	WQE[a]	NA
GRH FlowLabel	CA's set to 0; it will then be loaded with another value at the first encountered router. Alternately set according to application.	QP	QP	EE	WQE	WQE[a]	NA

Table 60 Packet Fields and Parameters by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
GRH Paylen	Length of the global packet; calculated by the transport based on the message length.	WQE	WQE	WQE	WQE	WQE[a]	NA
GRH NxtHdr	CA's set to IBA (0x1B)	Tr	Tr	Tr	Tr	WQE[a]	NA
GRH HopLmt	CA's set to 0; it will then be loaded with another value at the first encountered router. Alternately set according to application.	QP	QP	EE	WQE	WQE[a]	NA
GRH SGID	Source Global ID, from the port table and the index found in:	QP	QP	EE	WQE	WQE[a]	NA
GRH DGID	Destination Global ID	QP	QP	EE	WQE	WQE[a]	NA
BTH OpCode	Depends on operation, set by the transport layer.	Tr	Tr	Tr	Tr	NA	NA
BTH TVer	The version of the transport. This field depends on the revision of the device (0).	Tr	Tr	Tr	Tr	NA	NA
BTH P_Key	Partition Key, from the port table and the Index found in:	QP	QP	EE	QP	NA	NA
BTH DestQP	Destination QP *For RD mode responses, this is from the Request Packet Source QP as stored in the EEC	QP	QP	WQE*	WQE	NA	NA
BTH Pad	Length of packet pad; used to calculate actual data size. Calculated by the transport layer based on data size.	WQE	WQE	WQE	WQE	NA	NA
BTH SE	Solicited Event	WQE	WQE	WQE	WQE	NA	NA
BTH M	Migrate. Set by the transport dependent on the migration state.	Tr	Tr	Tr	Tr	NA	NA
BTH AckReq	Acknowledge request	Tr	0	Tr	0	NA	NA
BTH PSN	Packet Sequence Number	QP	QP	EE	QP	NA	NA
RDETH EEC	Destination EE Context	NA	NA	EE	NA	NA	NA
DETH Q_Key	Key which protects datagram QPs	NA	NA	WQE	WQE	NA	NA
DETH Source QP	Source QP. Set by transport for datagram services.	NA	NA	Tr	Tr	NA	NA
RETH	All fields of the RDMA Extended Transport Header (when used) are taken from the WQE	WQE	WQE	WQE	NA	NA	NA

Table 60 Packet Fields and Parameters by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
AtomicETH	All fields of the ATOMIC Extended Transport Header (when used) are taken from the WQE	WQE	NA	WQE	NA	NA	NA
AETH MSN	Message Sequence number (ACKs only)	QP	NA	0	NA	NA	NA
AETH Syndrome	Acknowledge syndrome, computed based on operation for reliable services	QP	NA	EE+ QP	NA	NA	NA
AETH RNR-NAK timer (TTTTT)	This value is placed in the AETH.TTTTT field when sending an RNR NAK. It denotes the minimum time to wait before retrying the request.	QP	NA	QP[b]	NA	NA	NA
AETH credit count (CCCCC)	This value is placed in the AETH.CCCCC field when sending an Ack in RC mode. It denotes the number of receive WQEs available to receive Send or RDMA write with immediate messages.	QP	NA	NA	NA	NA	NA
AtomicAckETH	ATOMIC data returned; the data is loaded as defined by the R_Key and Virtual Address, stored per WQE	WQE	NA	WQE	NA	NA	NA
Immediate data	Dependent on operation	WQE	WQE	WQE	WQE	NA	NA
Payload	Dependent on operation	WQE	WQE	WQE	WQE	WQE	WQE
ICRC	Calculated by transport; data dependent	link	link	link	link	NA	NA
VCRC	Calculated by Link layer; data dependent	link	link	link	link	link	link

a. Raw IP does not have a GRH, it has the similar looking IPv6 header. The Parameters are labeled GRH for convenience. This entire header is loaded from a data segment provided by the WQE.
b. The actual value of the TTTTT field depends on the reason for generating an RNR. For HCA's, when a WQE is not ready, this value comes from the QP. Otherwise, the value is determined by the implementor.

9.10.2 TRANSPORT CONNECTION PARAMETERS

The following are not sent "on the wire" but are needed to implement the protocol. This table is included to provide a better understanding of the parameters used by the transport layer to provide a connection. This list only covers elements mentioned in the IBA specification, other elements may be needed to completely implement connections.

Table 61 Connection Parameters by Transport Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
Connect state	State of connection (Reset, RTR, RTS, Error etc.)	QP	QP	EE+ QP	QP	QP	QP
Port number	Used only if there is more than a single port	QP	QP	EE	QP	QP	QP
Global/Local header	Determines if global header is to be attached or not.	QP	QP	EE	WQE	NA	NA
MTU	Size of the packets allowed on this connection.	QP	QP	EE	WQE	WQE	WQE
RNR NAK retry time	Time before performing a retry due to RNR; this is initialized by the AETH.TTTTT field in the RNR-NAK, and counts down from there.	QP	NA	QP or EE[a]	NA	NA	NA
RNR Retry init	Send Queue RNR retry count Initialization value	QP	NA	EE	NA	NA	NA
RNR Retry counter	Send Queue RNR Retry counter value	QP	NA	QP	NA	NA	NA
Local ACK Timeout	The exponent used to calculate the delay before an ACK is declared “lost”.	QP	NA	EE	NA	NA	NA
Error Retries	Send Queue retry count for sequence or time-out errors	QP	NA	EE	NA	NA	NA
MigState	Migration State (Migrated, Armed, ReArm)	QP	QP	EE	QP	NA	NA
Disable_E2E_Credits	Send queue use E2E protocol (depends on remote side’s ability to send credits)	QP	NA	NA	NA	NA	NA
Path Speed (IPD)	Controls packet emission for slower links	QP	QP	EE	WQE	WQE	WQE
PD	Protection Domain for this QP	QP	QP	QP	QP	QP	QP
RDD	Reliable Datagram Domain	NA	NA	QP+ EE	NA	NA	NA
XmitPSN	Sequence number used when sending	QP	QP	EE	QP	NA	NA
AckPSN	Sequence number expected for the ACKs	QP	NA	EE	NA	NA	NA
Rx ePSN	Sequence number expected when receiving	QP	QP	EE	NA	NA	NA
RxAckPSN	Number of unacknowledged Rx packets	QP	NA	EE	NA	NA	NA
SSN	Transmit messages Sent Sequence Number	QP	NA	NA	NA	NA	NA
Rx MSN	Message Sequence Number	QP	NA	NA	NA	NA	NA
Rx credits	Rx queue elements posted	QP	NA	NA	NA	NA	NA
LSN	Limit Sequence number (credit accounting)	QP	NA	NA	NA	NA	NA
SchQP_dequeue	QP at head of schedule queue (RD mode)	NA	NA	EE	NA	NA	NA
SchQP_enqueue	QP at tail of schedule queue (RD mode)	NA	NA	EE	NA	NA	NA

Table 61 Connection Parameters by Transport Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
SchQP_Next	Pointer to next QP to be scheduled (RD mode)	NA	NA	QP	NA	NA	NA
Num_RDMA_Reads	Number of RDMA READs or ATOMICs supported by remote side	QP	NA	1	NA	NA	NA
RDMAR/VA/R_Key/Size or ATOMIC “result”	The “hidden” stored address(s) of RDMA READ request(s) or ATOMIC results	QP	NA	EE	NA	NA	NA
RDMA PSN# or ATOMIC PSN #	Sequence number of requested op, used to match response on a repeat, and store reply PSN	QP	NA	EE	NA	NA	NA
RDMAR/ATOMIC Use	Usage of the resource; 1=RDMAR, 0=ATOMIC	QP	NA	EE	NA	NA	NA
Rx Completion Q		QP	QP	QP	QP	QP	QP
Tx Completion Q		QP	QP	QP	QP	QP	QP
Tx WQE pointer	Points to current Send WQE and its data segments for requests	QP	QP	QP	QP	QP	QP
Tx ACK WQE pointer	Points to current Send WQE and its data segments for Completions	QP	QP	QP	QP	QP	QP
Rx WQE pointers	Points to current Receive descriptor	QP	QP	QP	QP	QP	QP

a. This value is stored with the QP or EE at the implementor’s option; depending on whether the requester implements ’suspend’ for RNR-NAK.

9.10.3 PACKET HEADER AND DATA FIELD VALIDATION

The following tables provide an indication of the validation responsibility of the various header and data fields in the data packets for the various IBA services. The following terms are used in the table:

- Link — This indicates the value is checked by the link layer.
- Tr — This indicates that the value is checked against fixed values or used by the transport layer to select among choices.
- QP — This indicates that the value is checked against values from the QP context
- EE — This indicates that the value is checked against values from the EE context
- NC — The value is Not checked
- NA — Not Applicable
- WQE — The value is checked against information derived from the WQE

Table 62 Packet Fields Validation source by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
LRH VL	The VL on incoming packet.	link	link	link	link	link	link
LRH LVer	The version of the link level. This field depends on the revision of the device.	link	link	link	link	link	link
LRH SL	The SL to use for requests	NC	NC	NC	NC	NC	NC
LRH LNH IBA	IBA transport bit, indicates that BTH follows	Tr(1)	Tr(1)	Tr(1)	Tr(1)	Tr(0)	Tr(0)
LRH LNH GRH	GRH bit, indicates that a GRH follows	QP	QP	EE	Tr	Tr(1)	Tr(0)
LRH DLID	Destination local ID used for routing This is always checked at the link layer against Base LID and LMC.	link QP	link QP	link EE	link	link	link
LRH Packet Length	Length of the local packet; checked against PMTU at link, valid packet size at Transport, and data buffer size and protection values.	WQE	WQE	WQE	WQE	WQE	WQE
LRH SLID	Source local ID in ongoing packets.	QP	QP	EE	NC[a]	NC[a]	NC[a]
GRH IPVer”	Checked for the value ’6’	Tr	Tr	Tr	Tr[a]	Tr[ab]	NA
GRH Tclass	Traffic Class	NC	NC	NC	NC[a]	NC[ab]	NA
GRH FlowLabel	Flow label	NC	NC	NC	NC[a]	NC[ab]	NA
GRH Paylen	Length of the global packet; may be checked against PMTU and LRH Packet Length at link, valid packet size at Transport, and data buffer size and protection values.	WQE	WQE	WQE	WQE[a]	WQE[ab]	NA
GRH NxtHdr	Checked for the value 0x1B	Tr	Tr	Tr	Tr[a]	NC[ab]	NA
GRH HopLmt	Hop Limit	NC	NC	NC	NC[a]	NC[ab]	NA
GRH SGID	Source Global ID	QP	QP	EE	NC[a]	NC[b]	NA

Table 62 Packet Fields Validation source by Service

Parameter	Description	RC	UC	RD	UD	Raw IP	Raw ET
GRH DGID	Destination Global ID	QP	QP	EE	NC[a]	NC[ab]	NA
BTH OpCode	Depends on operation	Tr	Tr	Tr	Tr	NA	NA
BTH TVer	The version of the transport.	Tr	Tr	Tr	Tr	NA	NA
BTH P_Key	Partition Key; checked against the port partition table and an index in the:[c]	QP	QP	EE	QP	NA	NA
BTH DestQP	Destination QP; checked against the valid set and QP mode by transport.	Tr	Tr	Tr	Tr	NA	NA
BTH Pad	Length of packet pad; supplements LRH Packet Length.	WQE	WQE	WQE	WQE	NA	NA
BTH SE	Solicited Event; passed to upper layers for each message	Tr	Tr	Tr	Tr	NA	NA
BTH M	Migrate. Checked and used by transport to select alternate path parameters	Tr	Tr	Tr	NC	NA	NA
BTH AckReq	Acknowledge request	Tr	NC	Tr	NC	NA	NA
BTH PSN	Packet Sequence Number	QP	QP	EE	NC	NA	NA
RDETH EEC	Destination EE Context; checked against the valid set and EE mode by transport.	NA	NA	Tr	NA	NA	NA
DETH Q_Key	Key which protects datagram QPs	NA	NA	QP	QP	NA	NA
DETH Source QP	Source QP. Passed to upper layers for each message.	NA	NA	NC	NC	NA	NA
RETH	All fields of the RDMA Extended Transport Header (when used) are validated against protection parameters associated with QP state.	QP	QP	QP	NA	NA	NA
AtomicETH	All fields of the ATOMIC Extended Transport Header (when used) are validated against protection parameters associated with QP state.	QP	NA	QP	NA	NA	NA
AETH MSN	Message Sequence number (ACKs only)	QP	NA	Tr	NA	NA	NA
AETH Syndrome	Acknowledge syndrome	Tr	NA	Tr	NA	NA	NA
AtomicAckETH	Atomic data returned; Passed to upper layers for each message.	NC	NA	NC	NA	NA	NA
Immediate data	Dependent on operation; Passed to upper layers for each message.	NC	NC	NC	NC	NA	NA
Payload	Dependent on operation; Passed to upper layers for each message.	NC	NC	NC	NC	NC	NC
ICRC	Checked by transport	link	link	link	link	NA	NA
VCRC	Checked by Link layer; data dependent	link	link	link	link	link	link

a. For HCAs, this information is provided to upper layers.
b. Raw IP does not have a GRH, it has the similar looking IPv6 header. The Parameters are labeled GRH for convenience. This entire header is loaded from a data segment provided by the WQE
c. For QP1, the P_Key need only be a member of the port's Partition table, it is not checked against a QP index..

9.11 STATIC RATE CONTROL

As the traffic load increases in a fabric, resource contention increases. Congestion management is used to smooth operation, improve fabric efficiency, improve effective bandwidth, and improve average packet latency in the face of such contention.

There are a variety of mechanisms to address this problem. For this version of IBA, only static rate control will be defined. Future versions of the specification may provide definition of additional mechanisms.

Static rate control is the ability of an endnode to keep the rate of data sourcing into the fabric below a pre-configured value.

IBA supports Static Rate control in CAs to reduce congestion caused by a higher-speed CA injecting packets onto a path within a subnet at a rate that exceeds the path or destination CA's ability to sink. For example, a CA with a 10 Gbps link transmitting packets to a CA with a 2.5 Gbps link through an intermediate switch. In this case, the switch would be required to introduce "back-pressure" (limit the link-level flow control credits returned to the faster link) in order to prevent the slower link from being overrun.

IBA provides three mechanisms to manage static rate control.

- Device provided port rate information (see 16.3.3.1 ClassPortInfo on page 798)
- FM supported reporting of best possible rates for a source/destination pair (see 15.2.5.17 PathRecord on page 717).
- CA "Inter Packet delay" parameters in the connection setup MADs (described below)

9.11.1 STATIC RATE CONTROL FOR HETEROGENEOUS LINKS

A channel adapter has the ability to limit the rate of packets injected. This rate is based on the subnet-local destination port.

o9-163: If a port can support injection into the fabric at a rate greater than 2.5 Gbits/sec, this port shall provide static rate control as defined in this section.

The link rate supported is defined by the PortInfo:LinkWidthSupported and PortInfo:LinkSpeedSupported attributes. See Table 126 PortInfo on page 665 for a description of these.

o9-164: If a port is configured for injection into the fabric at a rate greater than 2.5 Gbits/sec, it shall not schedule a packet for injection into the local subnet until a programmable amount of time has passed since the last packet was scheduled for injection from this source port to the same destination port.

The link rate configured is defined by the PortInfo:LinkWidthActive and PortInfo:LinkSpeedActive attributes. See Table 126 PortInfo on page 665 for a description of these.

In the above, destination port refers to the full DLID (i.e. base DLID plus path bits) of the destination port within the local subnet, even for globally routed packets, while source port refers to the ingress port regardless of SLID (I.e. applies to all the SLIDs associated with this port).

The time to wait before transmitting a subsequent packet is based on the time it takes to transmit the current packet.

o9-165: If a port can support injection into the fabric at a rate greater than 2.5 Gbits/sec, the time to wait between scheduling packets destined for the same DLID and originating from the same port is determined by the Inter Packet Delay (IPD). Specifically, if a packet *b* is to be sent to the same DLID and using the same source port as packet *a*, then packet *b* shall not be scheduled until a time T_s has passed since packet *a* was scheduled, where T_s is calculated as: (IPD + 1) multiplied by the time it takes to transmit the first packet. Further, the time it takes to transmit a packet is calculated as LRH:PktLen*4/L_r where L_r is the port speed as obtained from the PortInfo:LinkWidthActive and PortInfo:LinkSpeedActive attributes.

The Inter Packet Delay (IPD) value is an 8-bit integer and is interpreted as depicted in the table below. Note that all 256 possible values are legal.

Table 63 Inter Packet Delay

IPD	rate	Comment
0	100%	Suited for matched links
1	50%	
2	33%	Suited for 30 Gbps to 10 Gbps conversion
3	25%	Suited for 10 Gbps to 2.5 Gbps conversion
11	8%	Suited for a 30 Gbps to 2.5 Gbps conversion

See 17.2.6 Static Rate Control on page 835 for which values of IPD CAs are required to support.

C9-226: If a CA is requested to use an unsupported value, the CA shall pick a supported value, and return that value in the appropriate MADs or verbs.

Each connected QP (EE context for RDs) should have a programmed IPD value. UDs should include the IPD in the WQE.

The same value of IPD should be programmed for each connected QP and WQE using the same port and same DLID. If different values of IPD are programmed, the CA may use any of these values for any of this traffic.

CHAPTER 10: SOFTWARE TRANSPORT INTERFACE

10.1 OVERVIEW

This chapter describes the software transport layer of the IBA. The software transport defines the capabilities and behavior of the Channel Interface (CI), the presentation of the channel to the Verbs Consumer. This interface is implemented as a combination of the Host Channel Adapter (HCA), its associated firmware, and host software. Specification of the API used by the Verbs Consumer to access the capabilities of the CI is outside of the scope of this architecture.

A concept frequently encountered in this specification is that of Verbs Consumer. The precise meaning of the phrase varies, as a function of context, but it always means, as defined in the Glossary, the executing entity employing the capabilities of the CI to accomplish some objective. In some instances the Verb Consumer may be a OS kernel thread, in others a user-level application, and in still others, some special, privileged process. Where the difference is important to the correct behavior of an implementation, it is defined explicitly, as in 11.1 Verbs Introduction and Overview on page 473; elsewhere, it is left unspecified.

While the Partitioning section is not strictly part of the software transport layer, it describes ideas that connect intimately with the semantics of the Queue Pair (QP), and are therefore reasonably elaborated in this chapter. In addition, giving the descriptions of the necessary entities here ensures their inclusion in the architecture specification.

10.1.1 INTRODUCTION

The CI is the locus of interaction between the consumer of IBA services and the instantiation of an IBA fabric. Access to the HCA is via Verbs, which enable creation and management of QPs, management of the HCA, and coping with error indications from the CI that may be surfaced via the Verbs. All these activities must be carried out so as to enable Verbs consumers to enjoy the same level of protection and security as are guaranteed other entities supported by the host operating system.

Fundamental to CI interaction is management of HCAs, which includes arranging access to them, accessing and modifying selected of their attributes, and shutting them down. These activities are described below, and details of the corresponding Verbs layer semantics are given in the next chapter.

Entities with central importance to CI operation are QPs. They must be created, associated with protection domains, modified as required, and destroyed to free up resources when no longer needed. In use, they provide repositories for addressing information needed by Verbs consumers, as well as protection information to guarantee the operational integrity of themselves and the host system. QPs provide for various modes of operation, depending upon the requirements of the consumer. Details of these modes, as well as the means of establishing them for a QP are described below, and corresponding Verbs semantics are detailed in the next chapter. For a graphical depiction of the QP, see Figure 11 on page 66.

As a central mode of QP operation, direct, protected access to consumer memory is critical to realizing the performance potential of the IBA. This chapter describes the semantics of memory access defined for the architecture, detailing the ideas of memory regions and windows, and their registration, access keys for local and remote access to registered memory, and the management of errors that may arise in this context.

A Work Request (WR) is an elementary object in the software transport layer, used by consumers to enqueue Work Queue Elements (WQEs) to the Send and Receive queues of a QP. The WQE is what identifies the individual events of communication over the IBA fabric. A graphical depiction of the WQE and QP can be seen in Figure 12 on page 67. The five varieties of WRs, and the dynamics of their creation, use, and disposition via entries in Completion Queues (CQEs) are described in the sections to follow, as are the disposition of errors that may arise as they are used. Details of their contents are given in the next chapter.

10.2 MANAGING HCA RESOURCES

10.2.1 HCA

Verbs allow the Consumer to open an HCA, retrieve HCA attributes, modify HCA attributes that can be changed by the Consumer, and close the HCA.

Queue Pairs, Completion Queues and other resources associated with a specific HCA instance cannot be shared across multiple HCAs, even if they are managed by the same device driver software.

The intent of the architecture is to allow an implementation to pass Work Requests and Completion Status to and from a user space Consumer process to the HCA without kernel involvement.

10.2.1.1 OPENING AN HCA

The Verb used to open an HCA returns an opaque object or handle to uniquely reference each HCA so that Consumers can distinguish between HCAs in the endnode.

Opening an HCA prepares the HCA for use by the Consumer. Once opened an HCA cannot be opened again until after it has been closed.

10.2.1.2 HCA ATTRIBUTES

HCA Attributes are device characteristics. These attributes must be able to be retrieved by the Consumer. The full list of HCA Attributes are defined in 11.2.1.2 Query HCA on page 476.

10.2.1.3 MODIFYING HCA ATTRIBUTES

Modification of a restricted set of HCA attributes is permitted. This is primarily restricted to performance and error counter management information. Most HCA Attributes are either fixed or manipulated through the Fabric Management Interface or General Services Interface.

10.2.1.4 CLOSING AN HCA

Close restores the HCA to its initialized condition, and deallocates any resources allocated during the HCA open.

It is not the responsibility of the Channel Interface to track any resources which were not allocated by the HCA open.

10.2.2 ADDRESSING

10.2.2.1 SOURCE ADDRESSING

For global addressing, each HCA Source Port has a GID Table containing the valid GIDs for the Source Port. The GID Table is obtained via the Query HCA Verb.

C10-1: For each HCA Source Port, the CI **shall** maintain a GID Table containing the valid GIDs for the Source Port.

Each Address Vector contains a Source GID Index. The Source GID Index specifies an index into the Source GID Table. The entry referenced by the Source GID Index defines the Source GID associated with the Address Vector.

C10-2: For each GID Table, the first entry in the table **shall** contain the read-only invariant value of the Base GUID.

For local addressing, IBA enables the use of multiple LIDs with each HCA port through the use of a Base LID, LMC and Path Bits.

C10-3: Each Address Vector **shall** contain specific Source Port LID Path Bits.

When the HCA constructs an SLID by taking the most significant bits from the port's Base LID and the least significant bits from an Address Vector's Path Bits, the port's LMC specifies how many bits come from the latter.

10.2.2.2 DESTINATION ADDRESSING

Addressing of destination endpoints is determined based on the Service Type of the QP:

C10-4: For Connection-oriented QPs, the destination address **shall** be stored in the QP Context, and **shall** be manipulated exclusively through the Modify Queue Pair Attributes Verbs.

o10-1: If the CI supports the RD Service, then for Reliable Datagram QPs, the destination address **shall** be stored in the EE Context, and **shall** be manipulated exclusively through the Modify EE Context Attributes Verb, and an EE Context **shall** be referenced by each individual Work Request (see 10.2.6 End-to-End Contexts).

o10-2: If the CI supports Raw Datagram Service, then for Raw QPs, the destination address **shall** be supplied via each individual Work Request.

C10-5: For Unreliable Datagram QPs, the destination address of the node **shall** be contained in an Address Handle, and an Address Handle **shall** be referenced by each individual Work Request.

An Address Handle is a consumer-opaque object that refers to a local or global destination. Verbs are used to create, modify and destroy Address Handles. Address handles are associated with protection domains. Protection domains are described in 9.11.2.3 Protection Domains.

C10-6: The CI **shall** support sending messages from a QP or EE addressed to the same or a different QP/EE on the same port in the sending HCA. Such messages **shall not** be transmitted through the fabric, but **shall** remain contained within that HCA.

No special addressing mechanisms are necessary to accomplish this; instead, the destination information in the source QP, EE, Address Vector, or Work Request is the same as that which any other node on the fabric would use to address the destination QP/EE.

10.2.2.3 PROTECTION DOMAINS

A Protection Domain (PD) is used to associate Queue Pairs with Memory Regions and Memory Windows, as a means for enabling and controlling HCA access to Host System memory. PDs are also used to associate Unreliable datagram queue pairs with Address Handles, as a means of controlling access to UD destinations. Queue Pairs are described in 10.2.3 Queue Pairs. Memory Regions and Memory Windows are described in detail in Section 10.6 Memory Management. PDs are specific to each HCA.

C10-7: Each Queue Pair in an HCA **shall** be associated with a single PD. Multiple Queue Pairs **shall** be able to be associated with the same PD.

C10-8: Each Memory Region, Memory Window, or Address Handle **shall** be associated with a single PD. Multiple Memory Regions, Memory Windows, or Address Handles **shall** be able to be associated with the same PD.

C10-9: Operations on a Queue Pair that access a Memory Region or a Memory Window **shall** be allowed only if the Queue Pair's PD matches the PD of the Memory Region or Memory Window.

C10-10: Operations on unreliable datagram queue pairs that access an Address Handle **shall** be allowed only if the Queue Pair's PD matches the PD of the Address Handle. If there is a mismatch, the Channel Interface **shall** return either Invalid Address Handle as an immediate error or Local Operation Error as a completion error.

10.2.2.4 Allocating a Protection Domain

Protection Domains are allocated through the Verbs.

A PD is required when creating a Queue Pair, registering a Memory Region, allocating a Memory Window, or creating an Address Handle.

A PD has no IB architected attributes. Operating Systems are commonly expected to enforce the policy that when a Verbs consumer creates a Queue Pair, registers a Memory Region, allocates a Memory Window, or allocates an Address Handle, it must specify a PD for association with the IB resources owned by it (that is, that were allocated by it).

10.2.2.5 Deallocating a Protection Domain

Protection Domains are deallocated through the Verbs.

C10-11: A PD **shall not** be deallocated if it is still associated with any Queue Pair, Memory Region, Memory Window, or Address Handle. If this is attempted, the Verbs **shall** return an immediate error.

10.2.3 Queue Pairs

The Verb consumer uses a Verb to submit a Work Request (WR) to a Send queue or a Receive queue. Associated Send and Receive queues are collectively called a Queue Pair (QP); these QPs drive the channel interface. A QP, which is a component of the channel interface, is not directly accessible by the Verbs consumer and can only be manipulated through the use of Verbs. See 10.8 Work Request Processing Model for a description of the WR submission process.

10.2.3.1 CREATING A QUEUE PAIR

Queue Pairs are created through the Verbs.

When a QP is created, a complete set of initial attributes must be specified by the Consumer. The attributes that need to be defined when the QP is created are denoted in 11.2.3.1 Create Queue Pair on page 486.

The maximum number of Work Queue Entries (WQEs) the Consumer expects to be outstanding on each work queue of the Queue Pair must be specified when the QP is created. The actual number of entries is returned through the Channel Interface for each work queue.

When setting the maximum number of outstanding work requests on a work queue, the consumer must take into account that this number must be large enough to encompass the number of work requests on that queue that have not completed plus the number of completed work requests for that queue that have not been freed through the associated CQ (see 10.8.5.1 Freed Resource Count on page 452). Note for unsignaled completions, the consumer cannot consider the work request completed until the work request has been confirmed completed as per 10.8.6 Unsignaled Completions on page 453.

10.2.3.2 QUEUE PAIR ATTRIBUTES

Queue Pairs have attributes that can be retrieved through the Query Queue Pair Verb. The complete list of QP Attributes is described in 11.2.3.3 Query Queue Pair on page 495.

10.2.3.3 MODIFYING QUEUE PAIR ATTRIBUTES

Certain QP Attributes may be modified after the QP has been created. The subset of QP Attributes which can be modified are defined in 11.2.3.2 Modify Queue Pair on page 488.

It is possible to modify the QP Attributes with Work Requests outstanding on the QP. Any Work Requests outstanding on the specified QP may not execute properly when the attributes are changed.

When setting the maximum number of outstanding work requests on a work queue, the consumer must take into account that this number must be large enough to encompass the number of work requests on that queue that have not completed plus the number of completed work requests for that queue that have not been freed through the associated CQ (see 10.8.5.1 Freed Resource Count). Note for unsignaled completions, the consumer cannot consider the work request completed until the work request has been confirmed completed as per 10.8.6 Unsignaled Completions.

A CI may support the ability to modify the maximum number of outstanding Work Requests on a QP. If it does so, it must be able to support it while Work Requests are outstanding. In addition, it must support resizing both work queues on every QP. If immediate errors are returned, the work queue(s) must be in the same state as it was prior to the attempt to resize the work queue(s). It is understood that this may adversely affect performance, but it must not be the cause of immediate, completion or asynchronous errors, with the exception of immediate errors returned by the Resize Queue Pair Verb. Note that a resize operation may adversely affect other QPs attempting to communicate with the QP during the resize operation in the form of timeouts and retries. It may also result in the loss of data in the form of dropped packets for unreliable service type QPs.

10.2.3.4 DESTROYING A QUEUE PAIR

Queue Pairs are destroyed through the Channel Interface.

When a QP is destroyed, any outstanding Work Requests are no longer considered to be in the scope of the Channel Interface. It is the responsibility of the Consumer to be able to clean up any associated resources.

Destruction of a QP releases any resources allocated below the Channel Interface on behalf of the QP. Outstanding Work Requests will not complete after this Verb returns.

10.2.3.5 SPECIAL QPS

QPs designated as special are the SMI QP (QP0), GSI QP (QP1) and the Raw IPv6 and Raw Ethertype QPs. These QP types are special because they have different and more restrictive properties than QPs designed for more general use.

Incoming messages to the SMI or GSI QPs may be consumed below the Verbs by a subnet management agent (SMA) or general services agent (GSA), respectively. If a MAD is consumed below the Verbs, the semantics must be consistent from the Verbs Consumer's point of view.

C10-12: Any message processing performed below the Verbs, e.g., by a SMA, **must not** disturb any WQEs and CQEs posted on behalf of the Verbs Consumer.

C10-13: The CI **shall** only allow direct access to the SMI and GSI QPs by privileged mode Consumers.

SMI and GSI QPs can share a completion queue, but neither can share one with any QP that is not of the SMI or GSI type.

Multiple Raw IPv6 and Raw Ethertype QPs are allowed on a single HCA. However, the demultiplexing algorithm that is applied to receiving messages between QPs is outside the scope of this specification.

10.2.4 Q_KEYS

A Queue Key (Q_Key) is a construct used in Datagram Service QPs to validate a remote sender's right to access a local Receive Queue. They are set through the Modify Queue Pair Verb, as well as within Work Requests. The values used for Q_Keys are not managed below the Verbs. Q_Keys are contained in the headers for IB Datagram packets.

Q_Keys have the following properties:

- A Q_Key must be established in the QP Context before Receive Descriptors can be posted to a QP.
- The Q_Key is a modifier in the Post Send Request Verb.

C10-14: For UD Service type QPs, the Q_Key in the QP Context **shall** be used to validate incoming packets. If the Q_Key does not match, the packet **shall** be silently dropped.

o10-3: If the CI supports the RD Service, then for RD QPs, the Q_Key in the QP Context **shall** be used to validate incoming packets. If the Q_Key does not match, the packet **shall** be NAK'd. This NAK **shall** result in the Send Queue at the remote node being placed into the appropriate error state as per the state diagram.

C10-15: A Q_Key **shall** be a modifier in the Post Send Request Verb for Datagram Service Type queue pairs. The Channel Interface **shall** examine the Q_Key in the Work Request. If the high order bit of the Q_Key is set, the outgoing packet **shall** contain the Q_Key from the QP Context. If the high order bit of the Q_Key is not set, the outgoing packet **shall** contain the Q_Key from the Work Request.

For the RD service class, Q_Key violation results in the source Send Queue transitioning to the error state. The destination has the option to support a Q_Key violation counter and trap. This counter may optionally be queried and set through the Queue Pair Verbs.

Q_Keys are not enforced on Raw QPs.

10.2.5 COMPLETION QUEUES

A CQ can be used to multiplex work completions from multiple work queues across queue pairs on the same HCA.

C10-16: The CI **shall** support Completion Queues (CQ) as the notification mechanism for Work Request completions. A CQ **shall** have zero or more work queue associations. Any CQ **shall** be able to service send queues, receive queues, or both. Work queues from multiple QPs **shall** be able to be associated with a single CQ.

10.2.5.1 Creating a Completion Queue

Completion Queues are created through the Channel Interface.

The maximum number of Completion Queue Entries (CQEs) the Consumer expects to be outstanding must be specified when the CQ is created. The actual number of entries is returned through the Channel Interface. It is the responsibility of the Consumer to ensure that the maximum number chosen is sufficient for the Consumer's operations; it must, in any case, arrange to handle an error resulting from CQ overflow.

C10-17: Overflow of the CQ **shall** be detected and reported by the CI before the next WC is retrieved from that CQ. This error **must** be reported as an affiliated asynchronous error -- see 11.6.3.2 Affiliated Asynchronous Errors on page 543.

10.2.5.2 Completion Queue Attributes

The only Completion Queue attribute is the maximum number of entries in the CQ. This attribute can be retrieved through the Query Completion Queue Verb.

Note that no Verb is provided which retrieves a CQ's set of associated Work Queues; the consumer is responsible for keeping track of this information, if needed.

10.2.5.3 Modifying Completion Queue Attributes

The CQ must be able to be resized through the Channel Interface while Work Requests are outstanding. It is understood that this may adversely affect performance, but it must not be the cause of immediate or completion errors, with the exception of immediate errors returned by the Resize Completion Queue Verb.

10.2.5.4 Destroying a Completion Queue

Completion Queues are destroyed through the Channel Interface.

C10-18: If the Verb to destroy a CQ is invoked while Work Queues are still associated with the CQ, the CI **shall** return an error and the CQ **shall** not be destroyed.

Destruction of a CQ releases any resources allocated below the Channel Interface on behalf of the CQ.

10.2.6 END-TO-END CONTEXTS

An End-to-End Context (EE Context) is a local HCA resource, used by the local HCA to track messages transferred between itself and another HCA, to support Reliable Datagram Service QPs. EE Contexts are established in an HCA by the Consumer through the Verbs.

o10-4: If the CI supports RD Service, the CI **shall** support an EE Context for use by the Consumer to provide the connection between two HCA ports. Each local EE Context **shall** be connected to exactly one destination EE Context.

The Consumer must establish a communication channel between a pair of EE Contexts, one on each HCA, before RD messaging can begin between the two HCAs. This communication channel must be established using the normal connection style semantics described in Chapter 12, Communication Services.

o10-5: If the CI supports RD Service, multiple connected EE Contexts (RD channels) **shall** be allowed between HCA port pairs. These EE Contexts are allowed to have either the same or different sets of attributes.

Any Work Requests outstanding on the specified EE Context may not execute properly when the attributes are changed.

The Consumer submits RD Work Requests to an RD type QP’s Send Queue. The Work Request specifies the EE Context to use to perform the actual message transfer. Work Requests may be submitted to a single RD QP that specify different EE Contexts as long as the EE Context specified is in the same RDD as the RD QP.

It is the responsibility of the Consumer to create, modify and destroy the EE Context, to use the Communication Services to gather the information necessary to transition the EE Context through the states as well as to fill out the necessary attributes for use.

It is important to note that Verbs manipulating EE Contexts, such as Create EE Context and Modify EE Context, use an EE Context handle, but Connection Management and submission of Work Requests to the Send Queue require the EE Context number. This number can be retrieved through the Query EE Context Verb.

10.2.6.1 CREATING AN EE CONTEXT

EE Contexts are created through the Channel Interface.

When an EE Context is created, a complete set of initial attributes must be specified by the Consumer. The attributes that need to be defined when the EE Context is created are denoted in Section 11.2.6.1 Create EE Context on page 503.

10.2.6.2 EE Context Attributes

EE Contexts have attributes that can be retrieved through the Query EE Context Verb.

The complete list of EE Context Attributes is described in Section 11.2.6.3 Query EE Context on page 507.

10.2.6.3 Modifying EE Context Attributes

Certain EE Context Attributes may be modified after the EE Context has been created. The subset of EE Context Attributes which can be modified are defined in Section 11.2.6.2 Modify EE Context Attributes on page 503.

It is possible to modify the EE Context Attributes when Work Requests requiring the EE Context are outstanding. Any outstanding WR which requires the specified EE context may not execute properly when the attributes are changed.

10.2.6.4 Destroying an EE Context

EE Contexts are destroyed through the Channel Interface.

When an EE Context is destroyed, any outstanding Work Requests which depend on the EE Context are expected to complete with an appropriate error.

Destruction of an EE Context releases any resources allocated below the Channel Interface on behalf of the EE Context.

10.2.7 Reliable Datagram Domains

A Reliable Datagram Domain (RDD) is a means to associate Queue Pairs with EE contexts.

o10-6: If the CI supports RD Service, each RD QP **shall** be associated with only one RDD. Multiple RD QPs **shall** be able to be associated with the same RDD.

o10-7: If the CI supports RD Service, each EE context **shall** be associated with only one RDD. Multiple EE contexts **shall** be able to be associated with the same RDD.

o10-8: If the CI supports RD Service, WRs which specify an EE Context on an RD Queue Pair **shall** be allowed only if the RDD in the EE Context matches the RDD in the QP. If the RDDs do not match, the initiator's work request will complete with a local operation error, with no effect on the destination's receive queue (see 11.6.2 Completion Return Status on page 540 for the correct error code).

o10-9: If the CI supports RD Service, the CI **shall** support at least two RDDs.

The purpose of defining the RDD construct is to ensure that it is possible reliably to separate user and kernel I/O RD traffic through an HCA. Note also that realizing that separation requires two EE contexts, as well.

10.2.7.1 ALLOCATING A RELIABLE DATAGRAM DOMAIN

Reliable datagram domains are allocated through the Channel Interface, using a privileged operation.

10.2.7.2 DEALLOCATING A RELIABLE DATAGRAM DOMAIN

Reliable datagram domains are deallocated through the Channel Interface.

o10-10: If the CI supports RD Service, an RDD **shall not** be deallocated if it is still associated with any Queue Pair or EE Context. If this is attempted, the CI **shall** return an immediate error.

10.2.8 INFINIBAND HEADER DATA AND SOURCES

The following table lists each of the data items present in an IBA protocol header, as well as some internal state needed to send packets. For each Transport Service Type, it lists the source of that data or state. The sources for each of these are accessed through the Verbs. This table is provided to establish the correlation between the packet fields and the constructs established through the Verbs. Note that the legend for Table 64 is Table 65, below.

.

Table 64 Packet Fields, Queue Parameters, and their Sources

Header	Field	RC	UC	RD	UD	Raw IP	Raw ET
LRH	Virtual lane	Computed from SL and CAP SL->VL table					
LRH	LRH version	Fixed					
LRH	Service level	QP		EE	AV	WR	
LRH	LRH next header – IBA transport bit	Fixed=1				Fixed=0	
LRH	LRH next header – GRH bit	QP		EE	AV	Fixed=1	Fixed=0
LRH	Destination local identifier	QP		EE	AV	WR	
LRH	Packet length	Computed from data/header length					

Table 64 Packet Fields, Queue Parameters, and their Sources (Continued)

Header	Field	RC	UC	RD	UD	Raw IP	Raw ET
LRH	Source local identifier (part not covered by LMC)	CAP					
LRH	Source local identifier (part covered by LMC)	QP		EE	AV	WR	
LRH	Reserved	Fixed=0					
GRH	IP version	Fixed=6				N/A	
GRH	Traffic class	QP		EE	AV	N/A	
GRH	Flow label	QP		EE	AV	N/A	
GRH	Payload length	Computed from data/header length				N/A	
GRH	Next header	Fixed				N/A	
GRH	Hop limit	QP		EE	AV	N/A	
GRH	Source GID	Computed from CAP table and index in QP		Com-puted from CAP table and index in EE	Com-puted from CAP table and index in AV	N/A	
GRH	Destination GID	QP		EE	AV	N/A	
BTH	OpCode	WR				N/A	
BTH	BTH version	Fixed=0				N/A	
BTH	Partition key	QP		EE	QP	N/A	
BTH	Destination queue pair	QP		WR		N/A	
BTH	Pad count	Computed from data & header length				N/A	
BTH	Solicited event	WR				N/A	
BTH	Packet sequence number	Computed from QP state		Com-puted from EE state	Com-puted from QP state	N/A	
BTH	Reserved	Fixed=0				N/A	
RDETH	Remote EE context	N/A		EE	N/A		
RDETH	Reserved	N/A		Fixed=0	N/A		
DETH	Queue key	N/A		WR or QP depend-ing on WR		N/A	
DETH	Source queue pair	N/A		QP		N/A	

Table 64 Packet Fields, Queue Parameters, and their Sources (Continued)

Header	Field	RC	UC	RD	UD	Raw IP	Raw ET
DETH	Reserved	Fixed=0				N/A	
RETH	Virtual address	WR			N/A		
RETH	R_Key	WR			N/A		
RETH	DMA length	WR			N/A		
AtomicETH	Virtual address	WR	N/A	WR	N/A		
AtomicETH	R_Key	WR	N/A	WR	N/A		
AtomicETH	Swap data	WR	N/A	WR	N/A		
AtomicETH	Compare data	WR	N/A	WR	N/A		
AETH	Message sequence number	Com-puted	N/A	Com-puted	N/A		
AETH	Syndrome	Com-puted	N/A	Com-puted	N/A		
RWH	Reserved	N/A					Fixed
RWH	EtherType	N/A					WR
AtomicAck-ETH	Original remote data	Memory	N/A	Memory	N/A		
	Local EE context	N/A		WR	N/A		
	Port number	QP		EE	QP		
	Transport Timeout	QP	N/A	EE	N/A		
	Retry count	QP	N/A	EE	N/A		
	RNR retry count	QP	N/A	EE	N/A		
	MTU	QP		EE	N/A		
	Maximum static rate	QP		EE	AV	WR	
	Protection domain	QP					
	Reliable datagram domain	N/A		QP/EE	N/A		
	Send PSN	QP		EE	N/A		
	Receive PSN	QP		EE	N/A		
	Outstanding atomics/RDMA reads supported at destination	QP	N/A	EE	N/A		
	Send CQ	QP					
	Receive CQ	QP					

The following table is the legend for the previous one.

Table 65 Legend for Table 64

Abbreviation	Meaning
AETH	Acknowledgement extended transport header
AtomicAck-ETH	Atomic acknowledgement extended transport header
AtomicETH	Atomic extended transport header
AV	Part of the address vector object defined by the Verbs
BTH	Base transport header
CA	Property of the channel adapter
CAP	Property of the channel adapter port
Computed...	Calculated from other values as specified
DETH	Datagram extended transport header
DS	Field taken from data segment pointed to by work request
EE	Taken from the EE context (RD service only)
Fixed	Value is determined by the specification and is the same in all packets.
GRH	Global routing header
LRH	Local routing header
Memory	Retrieved from host memory by CA
MTU	Maximum transfer unit
N/A	Not applicable to this Service Type
QP	Taken from Queue Pair state (the real QP in the case of RD service)
Raw	Raw Packet service
RC	Reliable Connected service
RD	Reliable Datagram service
RDETH	Reliable datagram extended transport header
RNR	Receiver not ready
RWH	Raw ethertype header
UC	Unreliable Connected service
UD	Unreliable Datagram service
WR	Taken from a Work Request

10.3 RESOURCE STATES

10.3.1 QUEUE PAIR AND EE CONTEXT STATES

This section contains the QP and EE Context state diagram. The same state diagram is used for both QPs and EE Contexts. This section will use the term QP/EE for this and, where differences are important, will note them. This section will contain a definition of the QP/EE states. The QP/EE states defined here and the transition order between the states are shown in Figure 122. EE Contexts are created only for the Reliable Datagram Service Type, whereas QPs are used for all Transport Service Types.

Note that while QPs and EE Contexts share the same state diagram, the EE Context state has no relationship to the states of the sending and receiving RD QPs using that EE Context. The reader should not assume that because a QP made a state transition that a EE Context associated with that RD QP will also transition, and vice versa.

Even though a subset of the states could occur in any order for some of the Transport Service Types, the states must transition in the order specified. This is to keep the state definitions consistent and error semantics simplified. The order chosen is that required to support the Reliable Connection Service Type, and to provide for completeness of the information needed to transfer data using an EE Context.

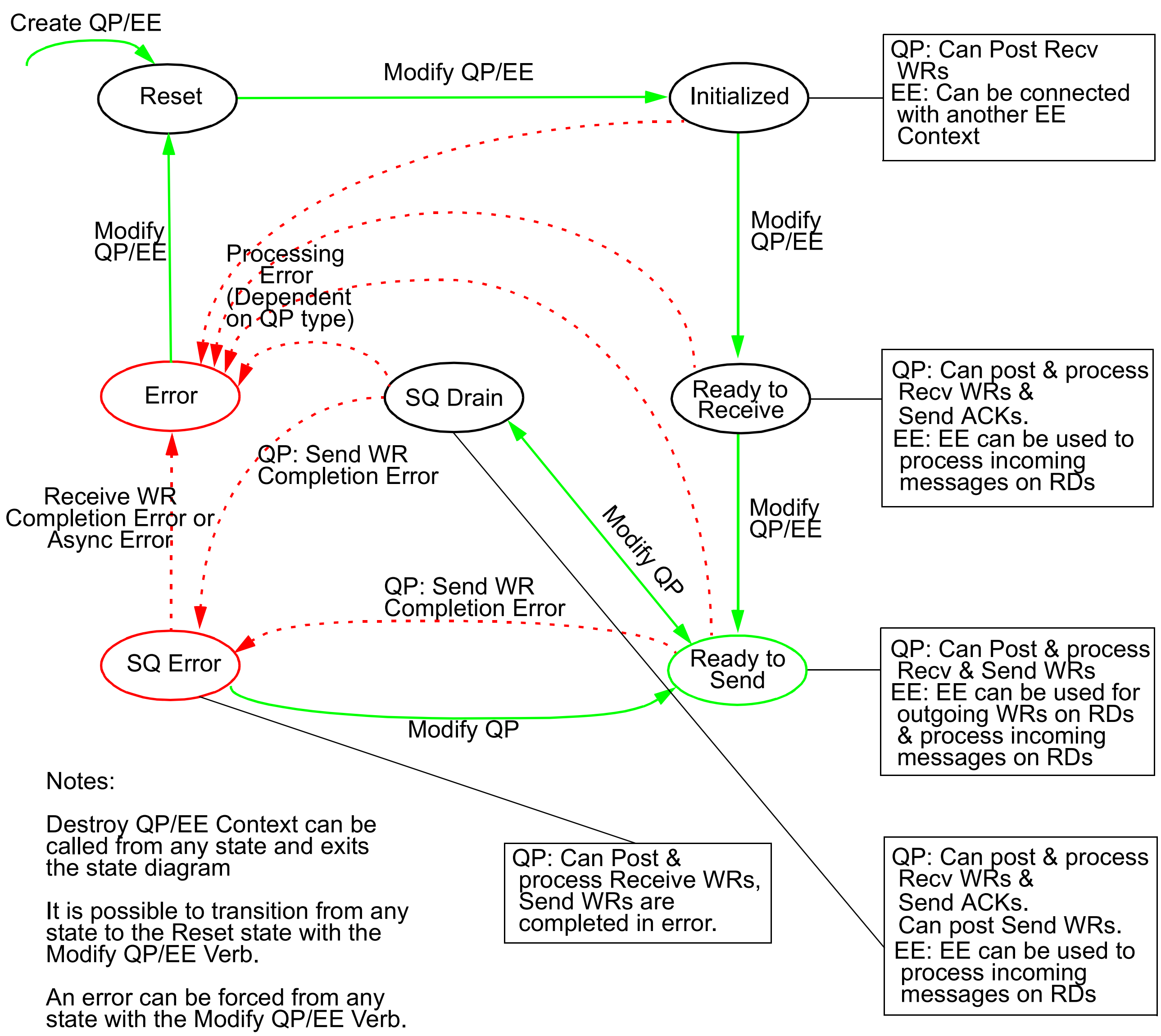


Figure 122 QP/EE Context State Diagram

It is important to understand that the QP/EE states are intended to be used as an integral part of the connection process. A thorough understanding of the connection process and the connection state diagram is assumed. This is discussed in detail in 12.9.7.1 Active States on page 576 and 12.9.7.2 Passive States on page 578.

C10-19: With one exception, the CI **shall** implement the Reset, Init, RTR, RTS, SQD, SQEr, and Error states for each QP. The CI **shall not** implement the SQEr state for RC QPs. Transitions between those states **must** be restricted to those shown in Figure 122.

o10-11: If the CI supports RD Service, the CI **shall** implement the Reset, Init, RTR, RTS, SQD, and Error states for each EE Context. Transitions between those states **must** be restricted to those shown in Figure 122.

10.3.1.1 RESET

The characteristics of the Reset state are:

C10-20: A newly created QP/EE **shall** be placed in the Reset state.

- It is possible to transition to the Reset state from any other state by specifying the Reset state when modifying the QP/EE attributes.
- Any resources required to implement the QP/EE have been allocated. For example, some implementations require WQEs and/or control structures to be allocated.
- The Modify Queue Pair and Modify EE Context Attributes Verbs are the only way for the Verbs Consumer to cause a transition out of the Reset State, without destroying the EE/QP.

C10-21: Upon creation, or transition to the Reset state, all QP/EE attributes **must** be set to the initialization defaults, as documented in the Create Queue Pair and Create EE Context Verbs.

- Transition out of the Reset state can be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- It is an error for a Work Request to specify an EE Context in the Reset state.

o10-12: If the CI supports RD Service, and a Work Request is submitted to the Send Queue of an RD QP specifying an EE Context in the Reset state, the Work Request **shall** be completed in error.

- No work requests can be outstanding which use an EE Context in this state.

o10-13: If the CI supports RD Service, any incoming messages which target an EE in the Reset state **must** be silently dropped.

For QPs:

C10-22: If a Work Request is submitted to a Work Queue while its corresponding QP is in the Reset State, an immediate error **shall** be returned.

- The Work Queues are empty. No Work Requests are outstanding on the work queues.
- All Work Queue processing is disabled

C10-23: Incoming messages which target a QP in the Reset state **must** be silently dropped.

10.3.1.2 INITIALIZED (INIT)

The characteristics for the Initialized state are:

- The basic QP/EE attributes have been configured as defined in Modify Queue Pair and Modify EE Context Attributes Verbs.
- Transition into this state is only possible from the Reset state.
- The Modify Queue Pair or Modify EE Context Attributes Verbs are the only way for the Verbs Consumer to cause a transition out of the Init state, without destroying the EE/QP.
- Transition out of the Init state can be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

o10-14: If the CI supports RD Service, any incoming messages which target an EE in the Init state **must** be silently dropped.

- It is an error for a Work Request to specify an EE Context in the Init state.

o10-15: If the CI supports RD Service, and a Work Request is submitted by the Consumer to the Send Queue of an RD QP specifying an EE Context in the Init state, the Work Request **shall** be completed in error.

For QPs:

- Work Requests may be submitted to the Receive Queue but incoming messages are not processed.

C10-24: The CI **shall** allow Work Requests to be submitted to a Receive Queue while its corresponding QP is in the Init State.

- It is an error to submit Work Requests to the Send Queue.

C10-25: If a Work Request is submitted to a Send Queue while its corresponding QP is in the Init State, an immediate error **shall** be returned.

- Work Queue processing on both queues is disabled.

C10-26: Incoming messages which target a QP in the Init state **must** be silently dropped.

10.3.1.3 READY TO RECEIVE (RTR)

The characteristics for the Ready to Receive state are:

C10-27: The CI **shall** support posting Work Requests to Receive Queue of a QP in the RTR state.

C10-28: Incoming messages targeted at a QP in the RTR state **shall** be processed normally.

o10-16: If the CI supports RD Service, and an incoming message is addressed to an EE Context in the RTR state, the message **shall** be processed normally.

- Transition into this state is possible only from the Init state, using the Modify Queue Pair or Modify EE Context Attributes Verbs.
- Transition out of the RTR state can be effected by calling the Destroy QP or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- It is an error for a Work Request to specify an EE Context in the RTR state.

o10-17: If the CI supports RD Service, and a Work Request is submitted by the Consumer to the Send Queue of an RD QP specifying an EE Context in the RTR state, the Work Request **shall** be completed in error.

For QPs:

- Work Queue processing on the Send Queue is disabled. It is an error to post Work Requests to the Send Queue.

C10-29: If a Work Request is submitted to a Send Queue while its corresponding QP is in the RTR State, an immediate error **shall** be returned.

10.3.1.4 READY TO SEND (RTS)

Before transitioning to this state, the QP/EE communication establishment protocol must be completed.

The characteristics for the Ready to Send state are:

- The channel between the requester's QP/EE and responder's QP/EE has been established for connected Service Types and RD channels.

- Transition into this state is possible only from the RTR and SQD states.
- The Modify Queue Pair or Modify EE Context Attributes Verbs are the only way for the Verbs Consumer to cause a transition out of the RTS state, without destroying the EE/QP.
- Transition out of the RTS state can be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

C10-30: The CI **shall** support posting Work Requests to a QP in the RTS state.

C10-31: Work Requests on a QP in the RTS state **shall** be processed normally.

C10-32: Incoming messages targeted at a QP in the RTS state **shall** be processed normally.

o10-18: If the CI supports RD Service, and an incoming or outgoing message utilizes an EE Context in the RTS state, the message **shall** be processed normally.

10.3.1.5 Send Queue Drain (SQD)

The characteristics for the Send Queue Drain state are:

C10-33: The CI **shall** support posting Work Requests to a QP in the SQD state.

C10-34: Incoming messages targeted at a QP in the SQD state **shall** be processed normally.

o10-19: If the CI supports RD Service, and an incoming message utilizes an EE Context in the SQD state, the message **shall** be processed normally.

- Transition into this state is possible only from the RTS state, using the Modify Queue Pair or Modify EE Context Attributes Verbs.

C10-35: When transitioning into the SQD state, the QP/EE's send logic **must** cease processing any additional messages. It **must** also complete any outstanding messages on a message boundary, and process any incoming acknowledgements. The CI **must not** begin processing additional messages which had not begun execution when the state transition occurred.

C10-36: When all expected acknowledgements have been received, and processing of send queue work requests has ceased, and if event notification has been requested, an Affiliated Asynchronous Event **shall** be generated.

- The consumer can use the asynchronous event to determine when a state transition is possible.
- It is possible to enter the RTS state or error states from the SQD state via Modify Queue Pair or Modify EE Context Attributes Verbs.
- Attributes may be modified during the transition from SQD to RTS, but both sides must have received the affiliated asynchronous event in order to safely change attributes.

o10-19.a1: If the CI supports the ability to change the physical port associated with an RC QP when transitioning from SQD to RTS, the CI shall associate the physical port, if a different physical port is specified, with the QP before the transition from SQD to RTS has completed. The physical port is an optional attribute in the Modify QP verb during the transition from SQD to RTS.

o10-19.a2: If the CI supports RD service and supports the ability to change the physical port associated with the EE Context when transitioning from SQD to RTS, the CI shall associate the physical port, if a different physical port is specified, with the EE Context before the transition from SQD to RTS has completed. The physical port is an optional attribute in the Modify EE Context verb during the transition from SQD to RTS.

- It is also possible to transition out of the SQD state by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- Work Queue processing on the Send side of the EE Context is disabled.

o10-20: If the CI supports RD Service, Work Requests submitted to the Send Queue of an RD QP, which specify an EE Context in the SQD state, **must** not be processed but **shall** remain enqueued.

- QPs associated with an EE do not transition to the SQD state automatically, nor is it inherently necessary they do so.

For QPs:

- Work Queue processing on the Send Queue is disabled.

C10-37: Work Requests submitted to the Send Queue of a QP in the SQD state **must not** be processed but **shall** remain enqueued.

10.3.1.6 SEND QUEUE ERROR (SQER)

The characteristics for the Send Queue Error state are:

- Transition into this state can only happen as the result of a Completion Error, which occurred during the processing of a Work Request on the Send Queue while in the RTS state.
- The transition into the Send Queue Error state applies to all QPs except for RC QPs.

C10-38: Receive Work Requests which were submitted to a Receive Queue prior to that queue's transition into the SQEr state **shall** continue to be processed normally. New Receives **must be able to** be posted to such a Receive Queue.

C10-39: A Work Request which caused the Completion Error leading to the transition into the SQEr state **must** return the correct Completion Error Code for the error through the Completion Queue.

- This WR may have been partially or fully executed, and thus may have affected the state of the receiver, as follows:

  Send operations may have been partially or fully completed; because of this, a completion queue entry may or may not have been generated on the receiver.

  RDMA Read operations may have been partially completed; because of this, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

  RDMA Write operations may have been partially completed; because of this, the contents of the memory locations pointed to by the remote address of their Work Requests are indeterminate. If the operation specified Immediate Data, a completion queue entry may or may not have been generated on the receiver.

  Atomic operations may, or may not have been attempted; because of this, the contents of the memory locations pointed to by the remote address of the Work Request may have a value consistent with either event. At the local node, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

C10-40: Work Requests on the Send Queue, subsequent to that which caused the Completion Error leading to the transition into the SQEr state, **must** return the Flush Error completion status through the Completion Queue.

- Depending on the Service Type of the QP, some of the subsequent WRs may have been in progress when the error occurred. This may have affected state on the remote node. The possible effects depend on the WR type as noted above.

- The Modify Queue Pair Verb can be used to transition from the SQEr state to the RTS state.
- The Modify Queue Pair Verb can be used to transition from the SQEr state to the Reset or the Error state.
- A Receive Queue Error or an Asynchronous Error will result in a transition to the Error State.
- Transition out of the SQEr state can be effected by calling the Destroy Queue Pair Verb, thus exiting the state diagram.

10.3.1.7 ERROR

The characteristics for the Error state are:

- Normal processing on the QP/EE is stopped.

C10-41: A Work Request which caused the Completion Error leading to the transition into the Error state **must** return the correct Completion Error Code for the error through the Completion Queue.

- This WR may have been partially or fully executed, and thus may have affected the state of the receiver, as follows:

  Send operations may have been partially or fully completed; because of this, a completion queue entry may or may not have been generated on the receiver.

  RDMA Read operations may have been partially completed; because of this, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

  RDMA Write operations may have been partially completed; because of this, the contents of the memory locations pointed to by the remote address of their Work Requests are indeterminate. If the operation specified Immediate Data, a completion queue entry may or may not have been generated on the receiver.

  Atomic operations may, or may not have been attempted; because of this, the contents of the memory locations pointed to by the remote address of the Work Request may have a value consistent with either event. At the local node, the contents of the memory locations pointed to by the data segments of their Work Requests are indeterminate.

C10-42: Work Requests subsequent to that which caused the Completion Error leading to the transition into the Error state, including those submitted after the transition, **must** return the Flush Error completion status through the Completion Queue.

- Depending on the Service Type of the QP, some of the subsequent WRs may have been in progress when the error occurred. This may have affected state on the remote node. The possible effects depend on the WR type as noted above.

- The Modify Queue Pair or Modify EE Context Attributes Verbs, specifying a transition to the Reset state, are the only means to effect a transition from the Error state to the Reset state.
- Transition out of the Error state can also be effected by calling the Destroy Queue Pair or Destroy EE Context Verbs, thus exiting the state diagram.

For EEs:

- If a Work Request is in process when the error occurred, the Work Request is completed with a completion error.

o10-21: If the CI supports RD Service, and an RD Work Request uses an EE context which is in the error state, that WR **must** be completed in error. This **shall** place the Sending QP into the SQEr state.

- Errors that occur on an EE may not have a corresponding effect on the QP state.

For QPs:

- For Affiliated Asynchronous Errors, it may not be possible to continue to process Work Requests. In this case, outstanding Work Requests will not be completed.
- When handling the error notification, it is the responsibility of the Consumer to ensure that all error processing has completed prior to forcing the QP to reset.

10.4 AUTOMATIC PATH MIGRATION

Automatic Path Migration is an optional facility that enables connection recovery in the case of failures. Automatic path migration is available for Reliable and Unreliable Connected QP Service Types and Reliable Datagram EE Contexts.

This section explains Automatic Path Migration from the software transport perspective. A hardware-centric description is contained in the Channel Adapter section, 17.2.8 Automatic Path Migration on page 836.

The Modify Queue Pair and Modify EE Context Attributes Verbs provide the basic capability to load an alternate path and to transition the path migration states defined in 10.4.1 Path Migration State Diagram.

Automatic path migration is enabled or re-enabled by loading an alternate path on the pair of connected QP or EE Contexts and setting the path migration state to Rearm. The Communication Manager defines protocols and mechanisms, which may be used to enable or re-enable Automatic Path Migration on both the local and the remote, connected QP or EE Context. The Communication Manager support for Automatic Path Migration is described in 12.6 Communication Management Messages on page 548 and in 12.8 Alternate Path Management on page 569.

Once Automatic Path Migration has been enabled on both ends of a connected QP/EE, it is possible for the migration to be initiated by transitioning the QP/EE path migration state from Armed to Migrated either from above or below the Verbs interface. The policy used by the Verbs Consumer or the CI to determine when a path migration should be attempted is outside the scope of the architecture.

10.4.1 PATH MIGRATION STATE DIAGRAM

o10-22: If Automatic Path Migration is supported, the CI **shall** implement the Migrated, Rearm, and Armed path migration states for each Reliable Connected and Unreliable Connected queue pair. Transitions between those path migration states **must** be restricted to those shown in Figure 123.

o10-23: If Automatic Path Migration and Reliable Datagram service are supported, the CI **shall** implement the Migrated, Rearm, and Armed path migration states for each EE Context. Transitions between those path migration states **must** be restricted to those shown in Figure 123.

The path migration states apply to a QP or EE Context, but are only tangentially related to the QP/EE Context states described in 10.3.1 Queue Pair and EE Context States.

o10-24: If Automatic Path Migration is supported, and the Verbs Consumer attempts to change the path migration state from Migrated to Rearm during a transition to a QP/EE state other than RTS, an immediate error **shall** be returned.

o10-25: If Automatic Path Migration is supported, and the Verbs Consumer attempts to change the path migration state from Armed to Migrated during a transition from a QP/EE state other than RTS or SQD, an immediate error **shall** be returned.

The relationship of the path migration states to the communication establishment process is defined in 12.9.7 State and Transition Definitions on page 576.

The path migration states are shown in Figure 123.

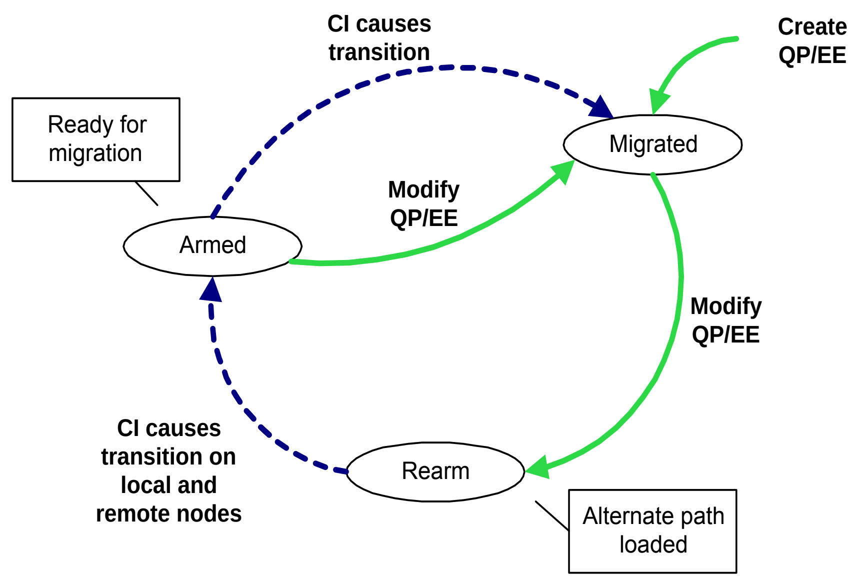


Figure 123 Path Migration State Diagram

10.4.1.1 MIGRATED

o10-26: If Automatic Path Migration is supported, the initial path migration state for a QP/EE **shall** be Migrated.

The Automatic Path Migration capability is suppressed while the state is set to Migrated.

The Verbs Consumer should leave the path migration state for the QP/EE to Migrated under the following circumstances:

- The local CI does not support Automatic Path Migration. If the Verbs Consumer attempts to change the path migration state using the Modify Queue Pair or Modify EE Context Attributes Verbs, an immediate error will be returned.
- The Verbs Consumer does not wish to enable Automatic Path Migration on the QP/EE pair.
- The remote CI does not support or desire Automatic Path Migration. If the Verbs Consumer changes the path migration state to Armed using the Modify Queue Pair or Modify EE Context Attributes Verbs, the path migration state for the QP/EE is changed

accordingly and no errors are generated. The local CI shall not transition the QP/EE from Rearm to Armed. Handling this condition is outside of the scope of the architecture.

The Verbs Consumer or the CI may set the path migration state to Migrated when the current path migration state is Armed and the QP/EE state is RTS. The decision of when to migrate is a matter of policy, which is outside the scope of the architecture.

o10-27: If Automatic Path Migration is supported, a transition from Armed to Migrated **shall** result in a migration to the alternate path on the local QP/EE. The CI **shall** raise the Path Migrated affiliated asynchronous event and **shall** send the next data packet using this QP/EE on the new path with a migration request.

The remote, connected QP/EE validates this request as defined in section 17.2.8 Automatic Path Migration on page 836.

o10-28: If Automatic Path Migration is supported, upon successfully validating an incoming packet's migration request, the CI **shall** set the path migration state for that QP/EE to Migrated, **shall** migrate to the alternate path, and **shall** also raise the Path Migrated affiliated asynchronous event for that QP/EE.

o10-29: If Automatic Path Migration is supported, upon failing to validate an incoming packet's migration request, the CI **shall not** modify the path migration state for that QP/EE, **shall not** migrate to the alternate path, but **shall** raise the Path Migration Request Failed affiliated asynchronous error for that QP/EE.

The Verbs Consumer should only set the path migration state to Migrated when the current path migration state is Armed and the QP/EE state is RTS. The Modify Queue Pair or Modify EE Context Attributes Verbs shall generate an immediate error when the Verbs Consumer attempts to set the path migration state to Migrated under any other condition.

o10-30: If Automatic Path Migration is supported, the CI **shall** only change the local path migration state to Migrated when the current state is Armed and the QP/EE state is RTS.

10.4.1.2 REARM

Only the Verbs Consumer is allowed to initiate the transition from Migrated to Rearm using the Modify Queue Pair or Modify EE Context Attributes Verbs.

o10-31: If Automatic Path Migration is supported, the CI **shall not** change the local path migration state from Migrated to Rearm except at the request of the Verbs Consumer.

The Verbs Consumer should load or reload the alternate path and ensure the remote node has accepted the alternate path prior to transitioning the state from Migrated to Rearm. A transition to the Rearm state indicates to the CI that the Verbs Consumer believes this QP/EE is ready to be transitioned to the Armed state. An invalid or stale alternate path will not generate any errors when the Verbs Consumer transitions the state to Rearm. Handling this condition is outside the scope of the architecture.

o10-32: If Automatic Path Migration is supported, a transition to the Rearm state **shall** cause the CI to attempt to coordinate with the remote, connected QP/EE to move both the local and the remote connected QP/EE into the Armed state in a lock-step manner.

The details regarding how the CIs perform this transition are contained in 17.2.8 Automatic Path Migration on page 836.

The QP/EEs at both ends of the connection must be in the Rearm state before the CI can transition them to the Armed state.

10.4.1.3 ARMED

The Armed state indicates that the CIs associated with the connected QP/EEs on both the local and the remote node are ready to perform a path migration.

10.5 MULTICAST SERVICES

Multicast is the ability to send a message to a single address and have it delivered to multiple queue pairs which may be on multiple endnodes. There are two types of multicast specified by IBA: IBA unreliable multicast, and raw packet multicast.

IBA Unreliable Multicast is an optional feature for HCAs. An HCA can be queried to determine the number of multicast groups supported by that HCA. The number of multicast groups is set to zero if the HCA does not support IBA unreliable multicast.

o10-33: If the CI supports IBA Unreliable Multicast, it **must** support at least one multicast group.

Raw packet multicast is an optional feature for HCAs. An HCA can be queried to determine whether it supports raw packet multicast.

10.5.1 MULTICAST GROUPS AND MULTICAST MESSAGE RECEPTION

A multicast group is a collection of endnodes which receive multicast messages sent to a single address. Multicast groups are a fabric management responsibility and are targeted through the use of an address.

10.5.1.1 IBA UNRELIABLE MULTICAST RECEPTION

o10-34: If the CI supports IBA Unreliable Multicast, a UD QP **must** be attached to a multicast group in order to receive IBA Multicast messages.

A QP is attached to or detached from a multicast group through the Verbs. The only function of the Attach QP to Multicast Verb is to assign a receive QP to the multicast group. If the HCA does not have the ability to allow the QP to attach to the multicast group, it shall return an immediate error indicating that there are insufficient resources.

One or more QPs, up to the maximum supported by the HCA, can be attached to each multicast group. In order to receive packets sent to the Multicast group, every QP attached to a particular multicast group should be a member of the same partition as the partition of the incoming packet.

Only Unreliable Datagram QPs can be used for IBA unreliable multicast. Therefore, all Unreliable Datagram semantics also apply to IBA unreliable multicast.

10.5.1.2 RAW PACKET MULTICAST RECEPTION

Raw packet QPs are not attached to multicast groups in order to receive raw packet multicast messages. If an HCA supports only one raw IPv6 QP per port, all raw IPv6 multicast messages received on a port are delivered to that port's raw IPv6 QP; if multiple raw IPv6 QPs are supported, raw IPv6 multicast messages are delivered to a subset of those QPs based on an implementation-defined policy which is outside the scope of IBA. Similarly, if an HCA supports only one raw ethertype QP per port, all raw ethertype multicast messages received on a port are delivered to that port's raw ethertype QP; otherwise, the distribution of those messages is again implementation-defined.

Only raw packet QPs can be used for raw packet multicast. Therefore, all raw semantics also apply to raw packet multicast.

10.5.2 MULTICAST WORK REQUESTS

10.5.2.1 IBA UNRELIABLE MULTICAST WORK REQUESTS

IBA unreliable multicast Work Requests must be submitted through the Post Send Request Verb to a single destination address. This destination address is specified with an Address Handle as part of the Work Request. Any Unreliable Datagram QP can be used to initiate an IBA unreliable multicast Work Request. A QP is not required to be attached to a Multicast Group in order to initiate an IBA Unreliable Multicast Work Request.

Send is the only operation allowed on an Unreliable Datagram Send Work Queue. Atomic and RDMA operations are not allowed. Unreliable Datagram messages must be no larger than the path MTU between the requester and the responder. Therefore, these restrictions apply to IBA unreliable multicast.

10.5.2.2 RAW PACKET MULTICAST WORK REQUESTS

Raw packet Multicast Work Requests must be submitted through the Post Send Request Verb to a single destination address. This destination address is specified as a modifier to the Post Send Request Verb. Any raw packet QP can be used to initiate a raw packet multicast Work Request.

Send is the only operation allowed on a raw packet Send Work Queue. Atomic and RDMA operations are not allowed. Raw packet messages must be no larger than the path MTU between the requester and the responder. Therefore, these restrictions apply to raw packet multicast.

10.5.3 MULTICAST DESTINATION ESTABLISHMENT

A multicast group is defined by a destination address. Multicast destination addresses have the same set of attributes as a unicast address.

o10-35: If the CI supports IBA Unreliable Multicast, then the CI shall drop all IBA Unreliable Multicast packets if the destination QP number is not 0xFFFFFF.

The special multicast QP number does not have to be the QP number used by the destination to receive a multicast.

C10-43: The method for preparing a multicast group address as a destination **shall be** the same as any other address specified in a Work Request on an Unreliable Datagram or Raw Packet Service Type.

Creating & Destroying multicast groups are fabric management issues. Permitting nodes to join and leave a multicast group is a fabric management issue. The MTU of a multicast group is the MTU specified when the multicast group is created and is a parameter in the multicast MAD.

o10-36: If the CI supports IBA Unreliable Multicast, then Multicast loopback, which is sending an IBA unreliable multicast message to a multicast group to which QPs within the sending node are attached, **must** be supported by the CI.

As with multicast reception, loopback for raw packet multicast depends on the number of raw packet queue pairs per port which an HCA supports. If an HCA supports only one raw queue pair of each type per port, then no loopback is performed; multicast messages sent on such a QP are not received on that same QP. If an HCA supports multiple raw QPs of each type per port, then multicast messages sent on one may or may not be received on another; the details of this are an implementation-defined policy which is outside the scope of IBA.

10.6 MEMORY MANAGEMENT

10.6.1 OVERVIEW

The InfiniBand™ Architecture provides sophisticated high performance operations like remote DMA and user mode IO. To achieve this goal, The InfiniBand™ Architecture has to specify appropriate memory management mechanisms. The overriding goals are performance, robustness and simplicity.

10.6.2 MEMORY REGISTRATION

An HCA, like a typical I/O bus host bridge, accesses Host System memory using what this specification refers to as physical memory addresses[1]. Physical address space for Host System memory is typically organized into pages of fixed or varying sizes, and a given logical data buffer that spans page boundaries usually has a non-contiguous physical address range.

Memory Registration provides mechanisms that allow Consumers to describe a set of virtually contiguous memory locations or a set of physically contiguous memory locations to the Channel Interface in order to allow the HCA to access them as a virtually contiguous buffer using Virtual Addresses.

All Consumers must explicitly register the memory locations containing data buffers before the HCA can access them.

C10-44: If a CI processes a WR or incoming RDMA or Atomic request that attempts to access memory locations that have not been registered, the CI **must not** perform the access, and the CI **must** return an appropriate error.

Registration may fail due to unavailability of the necessary Channel Interface resources. No memory is registered in this case.

C10-45: Registration **must** either fully succeed or fail in an atomic fashion.

10.6.2.1 MEMORY REGIONS

A set of memory locations that have been registered are referred to as a Memory Region.

The products of a memory registration operation are:

1. On some Host Systems, such "physical addresses" are actually mapped by the Host System memory controller to provide features such as memory interleaving or memory sparing, but this specification still refers to them as physical addresses.

- MemoryRegionHandle

  The Memory Registration Verbs produce a MemoryRegionHandle that is used to identify a specific Memory Region to the Memory Management Verbs.

- L_Key

  The Memory Registration Verbs produce an L_Key. The L_Key, along with a Virtual Address that is within the bounds of the region is used in a Work Requests's data segment to identify a memory location within a specific Memory Region to the CI.

- R_Key

  The Memory Registration Verbs produce, when requested, an R_Key. The R_Key, along with a Virtual Address that is within the bounds of the region is used in RDMA and Atomic operations to identify a memory location within a specific Memory Region to the CI.

- Virtual Address

  The Memory Registration Verbs are supplied (or in some cases produce) a Virtual Address that corresponds to the first memory location in the set of memory locations supplied to the Memory Registration Verbs.

When registering a Memory Region, the Consumer specifies whether Memory Windows are enabled to be bound to the Memory Region or not. Memory Windows are described in 10.6.6.2.2 Remote Access Through Memory Windows.

10.6.3 Access to Registered Memory

C10-46: The CI **shall** support the following access rights: Local Read, Local Write, Remote Read, and Remote Write.

o10-37: If the CI supports Atomic operations, the CI **shall** support the Remote Atomic access right.

10.6.3.1 Local Access to Registered Memory

A Memory Region is always accessible by the local HCA (i.e. a local HCA is an HCA in the same Host system as the Consumer) it was registered with, the type of access allowed depends on the Access Rights assigned to that Memory Region.

C10-47: The CI **shall** automatically include Local Read in every Memory Region's Access Rights.

The Consumer may request that Local Write be assigned to a Memory Region's Access Rights. If desired, policies related to preventing the assignment of Local Write to a Memory Region can be implemented by the Consumer.

10.6.3.2 REMOTE ACCESS TO REGISTERED MEMORY

The Consumer may, in addition to the Local access rights, assign Remote access rights to a Memory Region. Remote access rights are Remote Read, Remote Write and Remote Atomic. Remote access rights are individually selectable and when selected, allow one or more specific operation types to access the Memory Region. The Consumer is not allowed to assign Remote Write or Remote Atomic to a Memory Region that has not been assigned Local Write.

C10-48: If a Memory Registration specifies Remote Write or Remote Atomic without specifying Local Write, the CI **must** return an immediate error.

10.6.3.3 LOCAL ACCESS KEYS

When a set of memory locations are registered, an object called an L_Key, that is associated with that Memory Region is returned to the Consumer. Work Requests may require the Consumer to supply a locally accessible data buffer. Locally accessible data buffers are described by a Virtual Address that points to a location within a Memory Region, the L_Key associated with that Memory Region and the quantity of bytes in the buffer that may be used by the Work Request.

Memory Regions are described to the CI for local access by a combination of a Virtual Address within that Memory Region and the L_Key that was returned to the Consumer when the region was registered.

10.6.3.4 REMOTE ACCESS KEYS

When a memory region is registered with Remote Access Rights, an additional object called an R_Key, that is associated with that Memory Region is returned to the Consumer. Work Requests that will initiate an RDMA operation require the Consumer to supply a remotely accessible data buffer. Remotely accessible data buffers are described by a Virtual Address and an R_Key that have been supplied by the target endpoint. A Memory Region targeted by a remote operation must have appropriate Remote Access Rights for the type of operation.

Memory Regions are described to a CI for remote access by a combination of a Virtual Address within that Memory Region and the R_Key that was returned to the Consumer when the region was registered.

10.6.3.5 PROTECTION DOMAINS

A Protection Domain (PD) associates Memory Regions and Queue Pairs. Protection Domains are specific to each HCA. Each Memory Region must be associated with a single Protection Domain. Multiple Memory Regions may be associated with the same PD. Each Queue Pair in an HCA must be associated with a single Protection Domain. Multiple Queue Pairs may be associated with the same PD. Access to Memory Regions described in Work Requests and Remote Operation requests are allowed only when the Protection Domain of the Memory Region and of the Queue Pair that is processing the request are identical. The setting of protection domains is expected to be controlled by a Privileged Consumer.

10.6.3.6 SCOPE OF ACCESS

Memory is registered for use on a specific HCA. L_Keys and R_Keys are specific to an HCA and do not grant access to the Memory Region by other local HCAs. The CI is not required to enforce that L_Keys or R_Keys associated with one HCA will always result in an error if used with a different HCA.

10.6.3.7 MULTIPLE REGISTRATION OF MEMORY REGIONS

The same set of memory locations may be registered multiple times, resulting in multiple MemoryRegionHandles, L_Keys and R_Keys. Each Registration is considered a separate and distinct Memory Region and may be independently associated with a Protection Domain.

C10-49: The CI **shall** support independent registration of partially or completely overlapping sets of memory locations.

For cases where it's desired to have multiple registrations of a specific set of memory locations, provision for optimizing the use of Channel Interface resources is provided. See Section 11.2.7.7 Register Shared Memory Region on page 519.

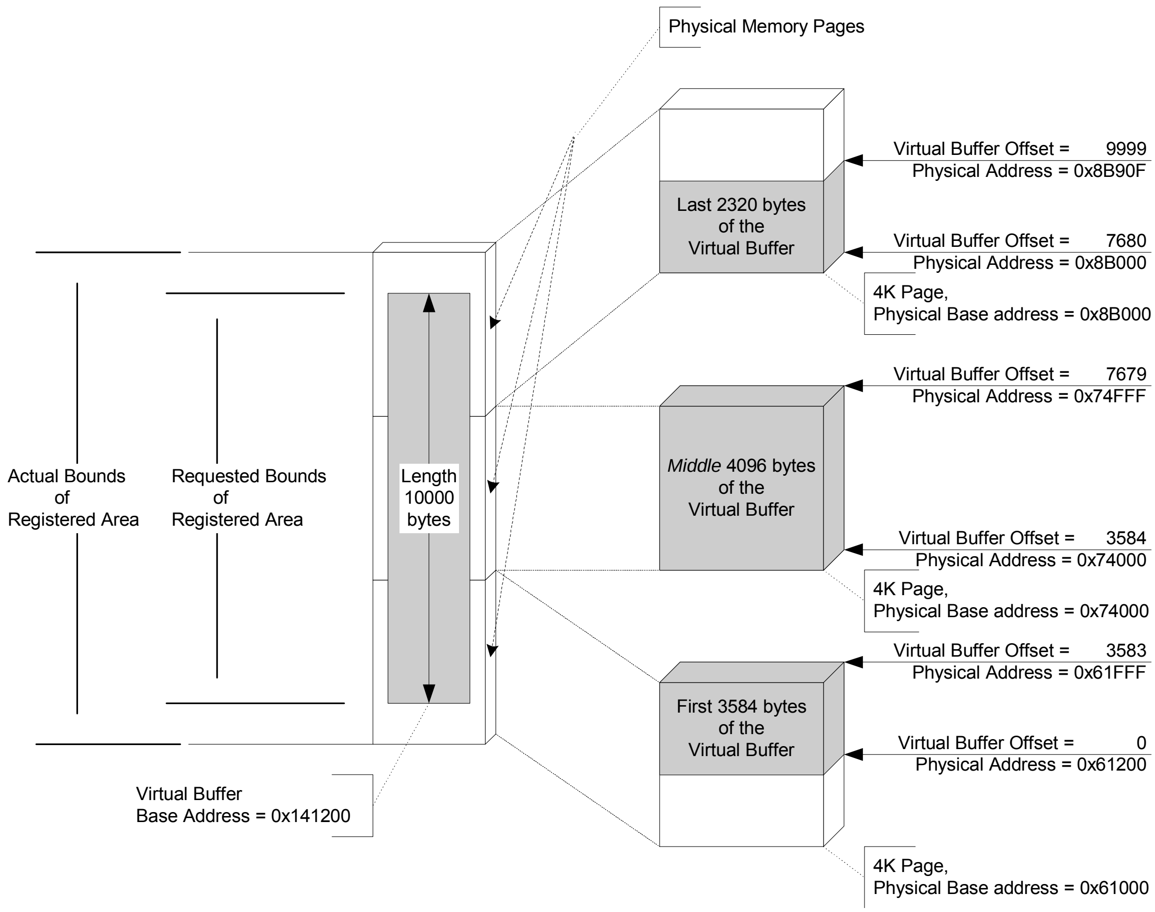
10.6.4 VIRTUAL ADDRESSES ("POINTERS")

Some processor architectures support global virtual address spaces of 80 bits or more. However, the virtual addresses ("pointers") most applications can readily manipulate and supply as parameters are typically either 32 bits or 64 bits, and actually serve as offsets into the handful of processor memory "segments" associated with the process. Thus, the virtual address parameters passed in by Consumers at the Verbs layer must each be interpreted in the proper context of their associated process. The L_Key or R_Key that accompanies each virtual address parameter helps the CI identify the appropriate context.

The virtual addresses ("pointers") that Consumers manipulate and pass as parameters are referred to simply as *Virtual Addresses* in this specification. The size of the Virtual Addresses used to specify a memory region to be registered and for local memory locations in Work Requests is implementation dependent. The size of Virtual Addresses used to specify remote memory locations in Work Requests is 64 bits.

10.6.4.1 VIRTUAL TO PHYSICAL TRANSLATIONS

Figure 124 Registered Virtual Buffer to Physical Page Relationship



10.6.4.2 REGISTRATION OF VIRTUALLY ADDRESSED REGIONS

A virtually contiguous set of memory locations are specified by a Virtual Address that points to the first byte of the set and the length of the set in bytes. Figure 124 illustrates a virtually contiguous set of memory locations backed by three physical pages. The size of the pages that back the region depend on the Host System hardware and Host Operating system.

C10-50: The CI **shall** support arbitrary byte alignment for the virtually contiguous buffer being registered.

C10-51: The CI **shall** support arbitrary length for the virtually contiguous buffer being registered, up to the limit specified by the HCA attribute.

The address translation and access rights of the region applies to each complete page within that memory region. The CI is not required to enforce access rights for local accesses with byte-level granularity.

The pages in the illustration are 4096 bytes each. The actual page size depends on the host hardware and host operating system.

In the example above, access to the memory locations at Virtual Addresses 0x141000 through 0x1411FF may be allowed even though they precede the first address of the region requested to be registered.

10.6.4.2.1 REGISTERED MEMORY RESIDENCY

C10-52: Using the Verbs defined in this specification, when a Memory Region is registered, every page within the region **must** be pinned down in physical memory.

This guarantees to the HCA that the Memory Region is physically resident (not paged out) and that the virtual to physical translation remains fixed while the region is registered.

The Verbs that register Virtually Addressed Regions are responsible for requesting that the OS pin the associated pages and for requesting from the OS any required per page Virtual to Physical translation information. The Channel Interface is not required to track pages common to Multiple registrations. The Channel Interface must be able to assume that the OS service that accepts requests for pinning and unpinning virtually addressed pages will maintain the appropriate reference counts on those pages such that pinned pages are not actually unpinned until the number of unpin requests equal the number of pin requests for any specific page.

The Verbs that register Virtually Addressed Regions are expected to request that the OS pin the pages associated with the region every time a region is registered regardless of any association with previously registered regions. The Channel Interface is not prohibited from implementing optimizations that reduce the number of OS service requests it makes for pinning and unpinning memory.

10.6.4.3 REGISTRATION OF PHYSICALLY ADDRESSED REGIONS

As an alternative to specifying a Region by a contiguous range in the Consumer's virtual address space mapped by the processor, Privileged Consumers can specify a Region by a list of physically addressed buffers, which correspond to pages mappable by the HCA. Besides the list of physical buffers, the Consumer supplies a requested “I/O Virtual Address” to be associated with the first byte of the Region, which is allowed to begin anywhere within the first physical buffer. The Consumer also supplies a byte offset that specifies where the Region begins within the first physical buffer. The Channel Interface returns the I/O Virtual Address that is actually assigned for the Region. The Channel Interface is not required to assign the I/O Virtual Address requested by the Consumer, but is encouraged to do so wherever possible.

The Consumer also supplies the length of the Region in bytes. The last byte of the Region, as specified by the Region length, must fall within the last physical buffer, but is allowed to fall anywhere within the last physical buffer.

The Virtual Address in this context is called an “I/O Virtual Address” since it isn’t necessarily mapped in the processor’s virtual address space, and might be used solely for local or remote accesses performed by the HCA.

The Maximum size of an I/O Virtual Address is 64 bits.

10.6.4.3.1 PHYSICAL BUFFER LISTS

Physical buffer lists used for registration consist of one or more physically contiguous memory regions that must start and end on an CI supported page boundary.

C10-53: If the physical buffer list in a physical memory registration contains an element that does not start and end on a CI-supported page boundary, the CI **shall** return an error.

All of the physical buffers in a physical buffer list must remain accessible by the CI until after the region has been deregistered.

For the case where the physical buffers in the physical buffer list are actually the pinned pages of a virtually addressed buffer, the Consumer is expected to keep those pages pinned while the region is registered.

It is the responsibility of the Consumer to determine if and when, after deregistration the pages should be unpinned. It is the responsibility of the Consumer to ensure proper operation in cases where the pages in the physically addressed region are also in use in a virtually addressed region that has been registered.

10.6.4.4 MEMORY REGION ERROR CHECKING

It is an error for a Consumer to use Virtual Addresses that are outside of the registered locations in a Memory Region.

10.6.4.4.1 ERROR CHECKING OF LOCAL ACCESSES TO MEMORY REGIONS

C10-54: The CI is **required** to ensure that the memory locations being referenced using a Virtual Address and L_Key are within a page of a Memory Region with the same PD as the QP that is processing the WR.

The CI is allowed to support finer-level granularity of local access control.

C10-55: The CI is **required** to ensure that the Local Access Rights of that Memory Region allow the type of access requested.

It is strongly encouraged that the Channel Interface check and ensure that the Virtual Address is within the Memory Region to which the L_Key is associated and report any bounds violation at access time. It is not mandatory that the Channel Interface enforce such checking.

10.6.4.4.2 ERROR CHECKING OF REMOTE ACCESSES TO MEMORY REGIONS

C10-56: The CI is **required** to ensure that the memory locations being referenced using a Virtual Address and R_Key are within a Memory Region with the same PD as the QP that is processing the Remote Operation. The CI **shall** enforce this with a granularity not to exceed 4096 bytes.

C10-57: The CI is **required** to ensure that the Remote Access Rights of that Memory Region allow the type of access requested.

It is strongly encouraged that the Channel Interface check and ensure that the Virtual Address is within the Memory Region to which the R_Key is associated and report any bounds violation at access time. It is not mandatory that the Channel Interface enforce such checking.

10.6.5 DEREGISTRATION OF REGIONS

When access to a Memory Region by a CI is no longer required, the Consumer may reverse the registration process for that region. The process of deregistering a Memory Region will revoke all HCA access rights to that Memory Region.

Memory locations that have been registered multiple times will be represented by multiple Memory Regions. The deregistration of single Memory Region prevents HCA access to those memory locations via the L_Key (and R_Key if any) associated with that Memory Region. Access to the memory locations via L_Keys and R_Keys associated with other Memory Regions is not affected.

C10-58: The CI **shall** support independent deregistration of partially or completely overlapping Registered Memory Regions.

C10-59: Work Requests or Remote Operation requests that are in process and actively referencing memory locations in a Memory Region that is deregistered **must** fail with a protection violation.

C10-60: Work Requests or Remote Operation requests that attempt to access memory locations in a Memory Region that has been deregistered **must** fail with a protection violation.

The Verbs that cause a Memory Region to be deregistered are expected to request that the OS unpin the pages associated with the region if a request to pin those pages was performed when the region was registered, regardless of any association with previously registered regions. The

Channel Interface is not prohibited from implementing optimizations that reduce the number of OS service requests it makes for pinning and unpinning memory.

10.6.6 MEMORY ACCESS CONTROL

The immediate Consumer of every memory registration related Verb is privileged code in the OS. In general, the OS is responsible for determining and enforcing access control policy for memory registrations it does on behalf of User-level Consumers. For instance, it is anticipated but not required that OSs will enforce policies similar to the following:

- A User-level Consumer has control over which of its memory areas can be accessed by HCA data transfer operations.
- A User-level Consumer can enable any local memory area it has access to for access by HCA data transfer operations.
- A User-level Consumer cannot enable HCA read access to memory areas that the Consumer itself doesn't have read access to.
- A User-level Consumer cannot enable HCA write access to memory areas that the Consumer itself doesn't have write access to.

When a Consumer creates QPs or CQs (through the appropriate Verbs), the HCA driver automatically allocates and pins any local memory needed for the associated control structures. Access by the HCA to these control structures is implicitly enabled. Access by the Consumer to these control structures is supported only indirectly through Verbs, and any Region Handles or L_Keys (if they exist) for the control structures are not exposed to the Consumer.

A Consumer controls which QPs can access which Memory Regions and which Memory Windows through the use of Protection Domains (PDs). Prior to creating any QPs, registering any Memory Regions, or allocating any Memory Windows, the Consumer will allocate one or more PDs. Then, when creating QPs, registering Memory Regions, or allocating Memory Windows, the Consumer specifies which PD each is associated with. QPs can only access Memory Regions or Memory Windows that are in the same PD.

10.6.6.1 LOCAL ACCESS CONTROL

With Sends and Receives, the Consumer explicitly specifies the buffers that are accessed through the local Data Segments it passes in the associated Work Requests. Each local Data Segment contains an address, its associated L_Key, and a length parameter. Multiple local Data Segments can be supplied for each send or receive where scatter/gather operation is desired.

Local Data Segments are also used for RDMA Write gather lists, RDMA Read scatter lists, and AtomicOp return values. Again each *local* Data Segment contains an L_Key which governs local access to the corresponding local Memory Region. However, the *remote* Data Segment associated with an RDMA Write, RDMA Read, or AtomicOp will contain an R_Key instead of an L_Key. This is discussed further below.

Two types of *local* access, read and write, are associated with Memory Regions. Send buffers and RDMA Write gather buffers require local read access. Receive buffers, RDMA Read scatter buffers, and AtomicOp return buffers require local write access.

Though memory registration is required to enforce local access only to page-level granularity, the local Data Segments used by Sends, Receives, RDMA Writes, RDMA Reads, and AtomicOps specify byte starting addresses and byte-count lengths. Thus the Consumer still has byte-level granularity of access control for local buffers accessed by these locally initiated operations. The Consumer can determine the actual range of access control enforced using the Query Memory *Region* Verb.

10.6.6.2 REMOTE ACCESS CONTROL

When a Consumer wants to allow remote agents to access its local memory using RDMA Writes, RDMA Reads, or AtomicOps, the Consumer must explicitly enable *remote* access and pass an appropriate *R_Key* to the remote agent for it to use when initiating these operations that target the Consumer's (local) memory.

A Consumer can use either of two mechanisms to enable remote access to its memory. The first mechanism involves enabling remote access when a Memory Region is registered. The second mechanism involves first allocating and then binding a Memory Window to an existing Memory Region. Either mechanism results in an R_Key with associated remote access rights for a specified memory area.

Three types of *remote* access — read, write, and atomic — are supported. RDMA Write requires write access at the remote target, RDMA Read requires read access at the remote target, and AtomicOps require atomic access at the remote target. While perhaps not obvious, it may make sense for a Consumer to allow atomic access but not allow write access, since AtomicOps are not required by the architecture to be atomic with respect to RDMA writes.

10.6.6.2.1 REMOTE ACCESS DIRECTLY WITH MEMORY REGIONS

When registering a Memory Region, a Privileged Consumer can generally specify any combination of remote access rights for the Region, including all or none. However, if a registration request does not specify local write access to the region, the CI will return an error if remote write or remote atomic access is specified.

If any remote access rights are specified, the Verb will return an R_Key. This R_Key grants the specified remote access rights for the entire Memory Region as bounded by the byte starting address and byte length, but the granularity of the access control actually enforced by the Channel Interface is allowed to be up to 4096 bytes. The Consumer can determine the actual range of access control enforced using the Query Memory *Region* Verb. It is strongly encouraged that the Channel Interface enforce access control with byte-level granularity.

10.6.6.2.2 Remote Access Through Memory Windows

When a Consumer needs more flexible control over remote access to its memory, the Consumer can use Memory Windows. Memory Windows are intended for situations where the Consumer:

- wants to grant and revoke remote access rights to a registered Region in a dynamic fashion with less of a performance penalty than using deregistration/registration or reregistration.
- wants to grant different remote access rights to different remote agents and/or grant those rights over different ranges within a registered Region.

To use a Memory Window, the Consumer allocates one and then binds it to a specified address range of an existing Memory Region that is enabled for use with Memory Windows. The range can include the entire Memory Region or any virtually contiguous subset of it. A Memory Window can only be bound to a Memory Region that belongs to the same Protection Domain.

C10-61: The CI **shall** enforce remote access control for Memory Windows with byte-level granularity.

When binding a Memory Window, a Consumer can request any combination of remote access rights for the Window. However, if the associated *Region* does not have local write access enabled and the Consumer requests remote write or remote atomic access for the *Window*, the Channel Interface must return an error either at bind time or access time. See 10.6.6.2.5 Error Checking at Window Bind Time and 10.6.6.2.6 Error Checking at Window Access Time.

C10-62: If a Memory Region does not have local write access enabled, the CI **shall** return an error if a Memory Window Bind request specifies remote write or remote atomic access to that Region. The CI **shall** allow all other requested access rights for Memory Windows.

A Consumer is allowed and commonly expected to enable remote access rights when binding a Window that it may not have enabled when it registered the underlying Region — provided it doesn't violate the above rule regarding local write access. For example, a Consumer might register a Region with no remote access rights, and later bind one or more Windows to that Region that obviously would grant remote access rights.

Allocating or *deallocating* a Memory Window requires a kernel transition, and thus incurs the associated software overhead. *Binding* a Memory Window is performed with a Work Request posted to a send queue, and thus incurs far less software overhead with typical implementations.

C10-63: Each time a given Memory Window is bound, the CI **shall** return an R_Key whose value is different from the immediate previous value. After the bind operation completes, any access attempts using the immediate previous R_Key **must** fail.

When the Memory Window is bound, the Verb returns the new R_Key immediately after posting the Work Request, even though the actual binding operation performed by the HCA hasn't yet occurred.

> **Implementation Note**: an envisioned implementation for an R_Key is to have it consist of two fields—an *index* field and a *key* field. The index field is used by the HCA to identify the associated Memory Window resource, and remains constant. The key field is changed each time the R_Key is bound, which guarantees that the immediate previous R_Key is invalidated as required. The use of a sufficient size key field and suitable random number with each binding can provide some amount of protection against the holder of an invalidated R_Key being able to access the Memory Window without authorization.
>
> The Channel Interface software that prepares the Bind Work Request generates the new key value and places it in the Work Request for the HCA to record in its Memory Window resource when processing the request. This way, the new R_Key value is fully determined and can be returned to the Consumer prior to the HCA processing the request.
>
> It is not required that Channel Interfaces use this implementation.

For correct operation, a Consumer must ensure that no remote agent attempts to use a new R_Key before its associated binding has been completed by the HCA. One technique to accomplish this is for the Consumer to submit the Bind operation to the same Send Queue it uses to send the message that conveys the new R_Key to the remote agent.

The Bind operation has a unique ordering rule:

C10-64: Any Work Request posted to a Send Queue subsequent to a Bind Work Request **shall not** begin execution until the Bind operation completes.

If the HCA detects an error with the Bind operation, it will put the QP into an error state. With the technique described earlier, the Bind operation is guaranteed to complete before the remote agent can possibly receive the new R_Key.

An envisioned common usage model is for a Memory Window to be allocated once and then used for multiple bindings. When a previously bound Memory Window is bound again, the previous R_Key and its associated bindings are automatically invalidated. Any remote agents needing to use the new Memory Window bindings must use the new R_Key.

If the Consumer wants to invalidate a Memory Window's bindings without deallocating the Window or enabling remote access to new areas, the Consumer can submit a Bind request specifying a length of zero.

C10-65: After a zero-length Memory Window Bind completes, the CI **shall not** allow any remote access to be performed to that Memory Window until a subsequent Bind re-enables remote access.

C10-66: The CI **shall** support multiple Windows bound to the same Memory Region, each with independent remote access rights, and their associated areas **shall** be allowed to be overlapping or disjoint.

10.6.6.2.3 REBINDING OR DEALLOCATING ACTIVE WINDOWS

Under normal operation, it is improper for a Consumer to deallocate or change the binding of a Memory Window while it is being accessed by a remote agent. However, this can occur if remote agents misbehave, or it can occur under error recovery circumstances.

C10-67: Any Remote Operation requests that are in process and actively using a Memory Window *when its binding is changed* **must** fail with a protection violation.

C10-68: Once the Bind operation has been reported to the Consumer as having completed, the Channel Interface **must** guarantee that no additional accesses can be performed under the immediate previous binding.

C10-69: Any Remote Operation requests that are in process and actively using a Memory Window *when it is deallocated* **must** fail with a protection violation.

C10-70: Once the Deallocate Memory Window Verb completes, the Channel Interface **must** guarantee that no additional accesses can be performed through that Memory Window while it remains deallocated.

10.6.6.2.4 DEREGISTERING REGIONS WITH BOUND WINDOWS

It is an error for a Consumer to deregister or reregister a Memory Region while it still has any Memory Windows bound to it. Such Windows are said to be “orphaned”. The Channel Interface must handle this error case as follows.

The Channel Interface is allowed to detect this error case and return an error without carrying out the deregister or reregister operation.

C10-71: If the CI allows a Memory Region deregister or reregister operation to create orphaned Windows, the CI **must** guarantee that any remote accesses attempted through the orphaned Windows will undergo the access checks and enforcement described in 10.6.6.2.6 Error Checking at Window Access Time.

10.6.6.2.5 ERROR CHECKING AT WINDOW BIND TIME

The following checks must be performed at Memory Window “bind time”, which is either when the Channel Interface is executing the Bind Memory Window Verb that prepares and queues the associated Work Request, or when the HCA is processing that Work Request.

C10-72: The Channel Interface **must** check and enforce that the Memory Window and QP belong to the same PD.

C10-73: The Channel Interface **must** check and enforce that Memory Windows are allowed to be bound to the specified Memory Region.

C10-74: The Channel Interface **must** check and enforce write permissions with the specified Memory Region, as described in 10.6.6.2.2 Remote Access Through Memory Windows .

C10-75: The Channel Interface **must** perform address bounds checks and PD checks with regard to the specified Memory Region.

10.6.6.2.6 ERROR CHECKING AT WINDOW ACCESS TIME

When the HCA processes an inbound RDMA or Atomic request that accesses a Window:

C10-76: The Channel Interface **must** check and enforce that the Memory Window and QP belong to the same PD.

C10-77: The Channel Interface **must** check and enforce the address bounds and access rights associated with the Window.

C10-78: The Channel Interface **must** check and enforce the access rights associated with each accessed page.

C10-79: For any previously undetected error cases where the Consumer orphaned the Window as described in 10.6.6.2.4 Deregistering Regions with Bound Windows, the Channel Interface **must** check and enforce that any pages accessed are in *some* Memory Region that belongs to the same PD as the Window.

The Channel Interface is not required to enforce that such pages are necessarily in the same Region to which the Window was bound. Again, it is strongly encouraged that the Channel Interface check and report these error cases at bind or deallocation time instead of access time.

10.7 WORK REQUESTS

Work Requests are used to submit units of work to the channel interface. There are different types of work requests supported and are abstracted throughout the Verbs.

How a work request targets its destination is dependent upon the work request and QP type. The target memory location is contained in the work requests's remote node address information (in the case of RDMA and Atomics) or in the remote receive QP WR's scatter/gather list (in the case of Send/Receive). The target QP depends on the QP type. Connected QPs have the destination QP contained in the local QP context. Datagram QPs have the destination QP contained as part of the work request. Raw QPs don't target a specific QP at the destination.

10.7.1 CREATING WORK REQUESTS

Work Requests are the only mechanism available to Consumers to generate work on work queues. Work requests are used only to pass the operation from the Consumer to the CI.

Work Requests are created by the Consumer above the Channel Interface using mechanisms provided by the OSV.

10.7.2 WORK REQUEST TYPES

There are five types of operations which may be posted to the Work Queues by the Consumer:

- Send/Receive
- RDMA Write
- RDMA Read
- Atomic Operations
- Bind Memory Window

Atomic Operations, Sends, RDMA operations and the Bind Memory Window operation may all take place on the same QP.

10.7.2.1 SEND/RECEIVE

C10-80: The CI **shall** support Send and Receive Operations on all Transport Service Types supported on the CI.

Sends must be posted to the Send Queue.

Receives must be posted to the Receive Queue.

C10-81: The responder's Receive QP **shall** consume a Work Request on reception of an incoming send message.

C10-82: The CI **shall** provide segmentation and reassembly for RC and UC Transport Service Types.

o10-38: If the CI supports RD Service, the CI **shall** provide segmentation and reassembly for RD.

10.7.2.2 RDMA

There are two types of RDMA: RDMA Read and RDMA Write.

RDMA Read Operations are supported only on the two reliable Transport Service Types—Reliable Connection and Reliable Datagram. RDMA Write Operations are supported on the two reliable Service Types plus the Unreliable Connection Service Type.

C10-83: The CI **shall** support RDMA Read Operations on the RC Transport Service Type.

C10-84: The CI **shall** support RDMA Write Operations on the RC and UC Transport Service Types.

o10-39: If the CI supports RD Service, the CI **shall** support both RDMA Read and Write Operations on the RD Transport Service Type.

RDMA Read and RDMA Write requests are submitted to the Send Queue.

C10-85: The responder's Receive Queue **shall not** consume a Work Request for an incoming RDMA Read.

C10-86: The responder's Receive Queue **shall** consume a Work Request when Immediate Data is specified in a successfully completed incoming RDMA Write.

C10-87: The responder's Receive Queue **shall not** consume a Work Request when Immediate Data is not specified in an incoming RDMA Write or the incoming RDMA Write was not successfully completed.

The target address of an RDMA request is the remote node's virtual address, a valid R_Key and length. The R_Key must be associated either a Memory Region or a Memory Window containing that virtual address.

Queue Pairs and Memory Regions or Memory Windows have RDMA Read attributes and RDMA Write attributes. These attributes are checked at the target end and are not checked at the source end.

C10-88: The CI **shall not** transfer data from an RDMA operation into the target memory unless the RDMA operation is enabled for the target QP.

10.7.2.3 ATOMIC OPERATIONS

IB Atomic Operations are architected as an optional feature to enable high-performance synchronization for distributed applications running on multiple hosts on the IB fabric.

Two operation types are supported: Compare & Swap and Fetch & Add. The operand size for these operations is 64 bits. It is the responsibility of the Channel Interface at the local endnode to do any transformation to match the endnode endian convention.

o10-40: If the CI supports Atomic operations, the CI **shall** support two types of Atomic operations, Compare & Swap and Fetch & Add.

o10-41: If the CI supports Atomic operations, the CI at the local endnode **shall** perform any byte ordering transformation required to match the endian endnode convention.

o10-42: If the CI supports Atomic operations, the CI **shall** implement Fetch & Add using two's complement arithmetic without saving the carry.

o10-43: If the CI supports Atomic operations, the CI **shall** return an error if the remote address of the Atomic operation is not aligned on a 64-bit boundary.

It is up to the Consumer to interpret whether the numbers are signed or unsigned.

Atomic Operations are supported only on the two reliable Transport Service Types—Reliable Connection and Reliable Datagram.

o10-44: If the CI supports Atomic operations, the CI **shall** support Atomic operations for the RC Transport Service Type.

o10-45: If the CI supports Atomic operations and the RD Transport Service Type, then the CI **shall** support Atomic operations for an RD QP.

o10-46: If the CI supports Atomic operations, the CI **shall not** support Atomic operations on any other Transport Service Types other than RC and RD.

Atomic Operation requests are posted to the Send Queue. The Atomic Operation request is made using the Post Send Request Verb. The results are contained in the data segment. The completion status of the request posted to the Send Queue indicates only if the Atomic Operation was successfully attempted. The Consumer must check the result to determine if a conditional operation took place.

o10-47: If the CI supports Atomic operations, the CI **shall** return the results of the operation in the Data Segment.

If an HCA supports atomics, then all atomic operation requests made to that HCA, referencing the same physical memory, are guaranteed to appear to be serialized with respect to each other. These operations may be directed at one or more queue pairs.

o10-48: If the CI supports Atomic operations, the CI **shall** provide the appearance that all Atomic operation requests made to the same HCA, referencing the same physical memory are serialized with respect to each other.

Atomic operation requests made to an HCA are not guaranteed to be serialized with respect to RDMA operation requests made to it or other HCAs in the system, or with respect to operations performed by other system components such as processors. Because of this behavior, if atomic operations on a particular area of memory are used to implement locks, all accesses to that memory must be done using atomic operations. In particular, it is not safe to use an RDMA read or Send/Receive to see if a lock is held, and it is not safe to use an RDMA write or Send/Receive to clear a lock.

Optionally, some systems may choose to provide a stronger guarantee: that all atomic operation requests made to all HCAs in the system, as well as all atomic operations performed on memory by other system components such as processors, referencing the same physical memory, are guaranteed to appear to be serialized with respect to each other. Again, these operations may be directed at one or several separate queue pairs. The definition of an "atomic operation" as performed by a system component which is not an HCA is implementation-dependent; for instance, a processor might be required to execute a particular instruction to produce an atomic operation.

o10-49: If the CI supports Atomic operations and the system provides Atomic access across the system, the CI **shall** provide the appearance that all Atomic operation requests that reference the same physical memory are serialized with respect to each other.

10.7.2.4 BIND MEMORY WINDOWS

The Bind Memory Window operation associates a previously allocated Memory Window to a specified address range within an existing Memory Region, along with a specified set of remote access privileges.

Bind Operations are supported only on the Reliable Connection, Unreliable Connection, and Reliable Datagram Service Types.

C10-89: The CI **shall** support Bind operations for RC and UC Transport Service Types.

o10-50: If the CI supports RD Service, the CI **shall** support Bind operations for the RD Transport Service Type.

Bind operations must be posted to the Send Queue. Binds affect only local HCA memory mapping resources and do not cause any packets to be issued over the link. No resources at the destination QP are affected.

10.7.3 WORK REQUEST CONTENTS

A Work Request contains all of the information required to perform the requested operation.

The contents of a Work Request for an operation posted to the Send Queue are described in Section 11.4.1.1 Post Send Request on page 525. The contents of a Work Request for an operation posted to the Receive Queue are described in Section 11.4.1.2 Post Receive Request on page 530.

10.7.3.1 SIGNALED COMPLETIONS

Work Requests always generate a Work Completion by default. This is referred to as a Signaled Completion. There is a mechanism where Work Requests posted to the Send Queue may not generate a Work Completion in the associated Completion Queue. This is referred to as an Unsignaled Completion. In order to use Unsignaled Completions, the QP has to be configured to support Unsignaled Completions and the Work Request must use the Signaling Indicator to request an Unsignaled Completion. Note that if a completion error occurs, a Work Completion will always be generated, even if the signaling indicator requests an Unsignaled Completion.

C10-90: The CI **shall** support both signaled and unsignaled completions.

C10-91: The CI **shall** generate a CQE when a Work Request completed under any of the following conditions:

- The Work Request completed in error.
- The Work Request was submitted to the Receive Queue.
- The Work Request was submitted to a Send Queue configured for only Signaled Completions.
- The Work Request was submitted to a Send Queue configured for Unsignaled Completions but the Work Request requested a Signaled Completion.

C10-92: The CI **shall not** generate a CQE when all of the following conditions have been met for a completed Work Request that was submitted to the Send Queue:

- The Send Queue has been configured to support Unsignaled Completions.
- The Work Request submitted to that Send Queue set the Signaling Indicator to request an Unsignaled Completion
- That Work Request completed successfully.

Work Requests using Unsignaled Completions can be determined to have been completed according to the rules in 10.8.6 Unsignaled Completions.

10.7.3.2 SCATTER/GATHER

A scatter/gather list may contain zero or more Data Segments. The buffers specified in a Work Request scatter/gather list must be registered with the Channel Interface prior to submission. These buffers must be considered to be in the scope of the Channel Interface from the time submitted to a work queue until completion of the Work Request has been confirmed. See 10.8.5 Returning Completed Work Requests and 10.8.6 Unsignaled Completions for a full description on when the completion of a Work Request is confirmed.

C10-93: If the total sum of all of the buffer lengths exceeds the maximum message payload size specified for an RC or UC QP, the CI **shall** report an error.

o10-51: If the CI supports RD Service, and if the total sum of all of the buffer lengths exceeds the maximum message payload size specified for an RD QP, the CI **shall** report an error.

A Data Segment is defined by a Virtual Address, L_Key and Length.

C10-94: The CI **shall** support scatter lists for Receive and RDMA Read operations.

C10-95: The CI **shall** support gather lists for Send and RDMA Write operations.

The order in which the Channel Interface accesses the memory described by a scatter/gather list is not defined by the architecture. In particular, this means that after completion of a Work Request whose scatter list contains overlapping Data Segments, the contents of the overlapped memory are undefined.

10.8 WORK REQUEST PROCESSING MODEL

10.8.1 OVERVIEW

The Work Request processing model describes how requests are submitted, processed by the HCA, and the results returned to the Consumer.

10.8.2 SUBMITTING WORK REQUESTS TO A WORK QUEUE

Work Requests are submitted to the HCA through the Verbs abstraction.

Work Queue Elements are abstract. This means that they are not accessible directly by the Consumer of the Channel Interface.

The intent of the architecture is to allow an implementation to pass Work Requests from a User-level Consumer process to the HCA without kernel involvement.

The QP can accept Work Requests only when the QPs are in states that allow them to be submitted. The rules are as follows:

C10-96: The QP **shall** process Work Requests submitted to the Send Queue as described in the rules that follow:

- Return an immediate error if the QP is in the Reset, Init and RTR states.
- Are processed when the QP is in the RTS state.
- Are completed in error, assuming that processing is able to continue when the QP is in the SQEr or Error state.
- Are enqueued but not processed when the QP is in the SQD state.

C10-97: The QP **shall** process Work Requests submitted to the Receive Queue as described in the rules that follow:

- Return an immediate error if the QP is in the Reset state.
- Are accepted, but incoming messages are not processed when the QP is in the Init state.
- Are processed when incoming messages arrive and the QP is in the RTR, RTS, SQD, or SQEr state.
- Are completed in error, assuming that processing is able to continue when the QP is in the Error state.

The modifiers in the Work Request are instantiated into the next free WQE in the specified Work Queue and the CI is informed that a new WQE has been added to the queue.

Figure 125 shows the transformation of a Work Request into a WQE to be processed by the HCA.

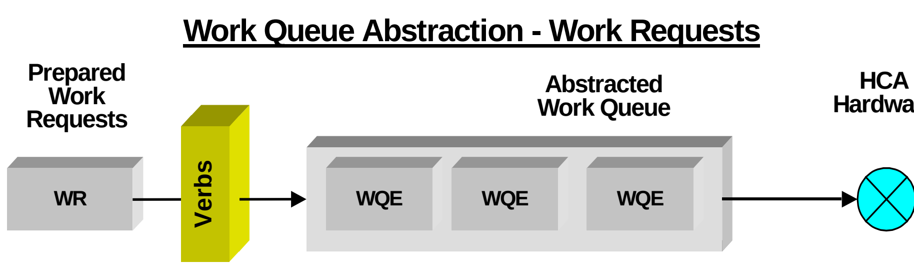


Figure 125 Work Queue Abstraction

10.8.3 WORK REQUEST PROCESSING

Processing of Work Requests submitted to a Work Queue are initiated in the order submitted. There is no ordering between WRs submitted to the send queue and WRs submitted to the receive queue. Send WRs are initiated in the same order they were passed to the Verbs layer with respect to other sends WRs submitted to the same send queue. Likewise, receive WRs are initiated in the same order they were passed to the Verbs layer with respect to other receive WRs submitted to the same receive queue.

C10-98: The CI **shall** initiate Work Requests submitted to a single queue in the order in which those Work Requests were submitted to that queue.

C10-99: For all Service types except RD, Work Requests submitted to the same Receive Queue **shall** complete in the same order in which they were submitted.

Resources associated with a Work Request must be considered to be in the scope of the Channel Interface from the time the Work Request is submitted to a Work Queue until the completion for that Work Request has been confirmed. See 10.8.5 Returning Completed Work Requests and 10.8.6 Unsignaled Completions for a description of when a Work Request completion is confirmed.

Work Requests submitted to a single Work Queue complete in the same order as the requests were submitted, according to the Ordering Rules.

The exception to this rule is that reliable datagrams are permitted to complete out of order on the Receive Queue.

- Reliable datagrams originating from a specific Send Queue complete in the same order they were submitted when they are sent to the same Receive Queue.

o10-52: If the CI supports RD Service, Work Requests submitted to the same RD Send Queue **shall** complete in the same order in which they were submitted.

o10-53: If the CI supports RD Service, Work Requests submitted to the same RD Receive Queue **shall** complete in the same order in which they were submitted when the requests originate from the same remote RD Send Queue.

Receive completions from reliable datagrams sent from multiple Send Queues are allowed to be interleaved on the Receive Queue.

As shown in Table 66 Work Request Operation Ordering, ordering semantics for WRs submitted to the Send Queue vary according to the operation type. Some operations can begin processing within the CI while other operations are still outstanding, potentially yielding out-of-order semantics for certain operation sequences. For cases enumerated below, in-order semantics can be guaranteed by setting the Fence Indicator for appropriate WRs. When the Fence Indicator is set for a given WR, that WR cannot begin to be processed until all prior RDMA Read and Atomic operations on the same Send Queue have completed.

C10-100: When the Fence Indicator has been set in a Work Request, the Send Queue **shall not** begin processing that Work Request until all prior RDMA Read and Atomic Operations on that Send Queue have completed.

Here are the cases where the Fence Indicator can be used to guarantee in-order semantics:

- An RDMA Read won't necessarily complete before subsequent Sends, RDMA Writes, or Atomics are initiated and observed by the target. If the target Consumer then modifies memory locations being returned by the RDMA read, the RDMA read could return the newly modified data instead of the original data. Setting the Fence Indicator for the subsequent operation in each case guarantees that the operation will not be observed by the target until all prior RDMA Reads complete.

- An RDMA Read can return data that's been modified by subsequent Sends, RDMA Writes, or Atomics if they target memory locations being returned by the RDMA Read. Setting the Fence Indicator on the subsequent operation in each case guarantees that the operation will not affect data being returned by a prior RDMA read.
- RDMA Read or Atomic operations won't necessarily complete before subsequent Sends, RDMA Writes, or Atomics are initiated and observed by the target. If one of the former operations completes in error on the initiator side because its ACKs fail to return successfully, the subsequent operation could still be observed by the target, and the target Consumer might take some undesired action. Setting the Fence Indicator on the subsequent operation in each case guarantees that it can't be observed by the target unless all prior RDMA Reads and Atomics complete successfully on the initiator side.

The Bind operation has a unique ordering rule: any Work Request posted to a Send Queue subsequent to a Bind must not begin execution until the Bind operation completes. However, note that a Bind operation itself can begin execution in some cases before prior operations have necessarily completed.

Ordering guarantees for processing and completion notifications exist only between Work Requests submitted to the same queue. The ordering across multiple Work Queues is undefined.

C10-101: The CI **shall** provide the guarantees for processing and completion notifications between Work Requests submitted to the same Send Queue as specified by the ordering rules in Table 66.

Ordering Rules:

- Receive Queues are FIFO queues with the exception of the reliable datagram issue described above.
- Send Queues are FIFO queues, according to the rules in Table 66 Work Request Operation Ordering. The Fence Indicator can be used to require strict ordering.

Table 66 Work Request Operation Ordering

First Operation	Second Operation: Send	Bind Window	RDMA Write	RDMA Read	Atomic Op
Send	#	#	#	#	#
Bind Window	#	#	#	#	#
RDMA Write	#	#	#	#	#
RDMA Read	F	F	F	#	F
Atomic Op	F	F	F	#	F

Table 67 Ordering Rules Key

Symbol	Description
#	Order is always maintained.
F	Order maintained only if second operation has Fence Indicator set

10.8.4 COMPLETION PROCESSING

The results from a Work Request operation are placed in a Completion Queue Entry (CQE) on the CQ associated with the Work Queue when the request has completed.

A CQE must be generated before a Work Completion can be returned to the Consumer. Note that not all Work Requests will generate a completion, due to unsignaled completions. The rules for when a CQE is generated are outlined in 10.8.5 Returning Completed Work Requests.

C10-102: For completed Work Requests that generate a Work Completion, the CI **shall** place that Work Completion on the CQ associated with the Work Queue.

A CQE is an internal representation of the Work Completion.

10.8.5 RETURNING COMPLETED WORK REQUESTS

All completions are abstracted through the Verbs. The only method of retrieving a Work Completion is through the Verbs.

Completions are always returned in the order submitted to a given work queue with respect to other Work Requests on that work queue. Ordering rules of completion entries from multiple work queues associated with a given completion queue are not mandated by this specification.

A retrieved Work Completion is no longer in the domain of the Channel Interface. Therefore, a Work Completion can only be retrieved once.

C10-103: The CI **shall not** allow a specific Work Completion to be retrieved more than once.

The Work Completion contents are specified in 11.4.2.1 Poll for Completion on page 531.

A Consumer can find out when a Work Completion can be retrieved through polling or notification.

C10-104: The CI **shall** return a Work Completion for a Work Request that completed with a signaled completion.

C10-105: The CI **shall** return a Work Completion for a Work Request submitted to a Send Queue that completed in error.

C10-106: The CI **shall** return a Work Completion for the completion of a Work Request submitted to a Receive Queue.

A Work Request is confirmed when the associated Work Completion is retrieved from its CQ.

C10-107: The CI **shall not** access any buffers associated with the Work Request once the associated Work Completion has been retrieved.

10.8.5.1 FREED RESOURCE COUNT

One of the modifiers returned with the completion is a count that informs the Consumer of the number of work request resources freed by this completion. This applies only to Reliable Datagram Receive Queues. Work request resources refers to Channel Interface resources allocated on behalf of the Consumer, such as available WQEs for a given Work Queue, and not direct Consumer resources, such as buffers.

If this count is zero, this indicates that no receive queue work queue elements have been freed when this Work Completion was generated. If this count is greater than zero, the Consumer can assume that the counter indicates the number of work requests released from the RD RQ. This is useful for the Consumer to keep track of the number of available work requests which can be outstanding.

Buffers associated with the outstanding work request associated with this work completion are no longer considered to be in the scope of the HCA, regardless of the Freed Resource Count.

For most implementations, this count is expected to be one with every work completion.

o10-54: If the CI supports RD Service, when a Work Completion associated with a Work Request posted to an RD RQ is retrieved, the CI **shall** return a count of the number of Work Request resources freed through the Verbs.

10.8.6 Unsignaled Completions

An unsignaled Work Request that completed successfully is confirmed when all of the following rules are met:

- A Work Completion is retrieved from the same CQ that is associated with the Send Queue to which the unsignaled Work Request was submitted.
- That Work Completion corresponds to a subsequent Work Request on the same Send Queue as the unsignaled Work Request.

C10-108: The CI **shall not** access buffers associated with an Unsignaled Work Request once a Work Completion has been retrieved that corresponds to a subsequent Work Request on the same Send Queue.

10.8.7 Asynchronous Completion Notification

The Consumer may register a completion notification routine to be called when a new entry is added to the CQ using the Set Completion Event Handler Verb.

C10-109: A CI **shall** support registering a single CQ Event Handler per HCA.

C10-110: The CI **shall** replace any previous handler with the handler specified in a new Request Completion Event Verb invocation.

The Request Completion Event Verb is set on a CQ basis. This is a one-shot notification; at most, one notification will be generated per call to this Verb. Once CQ notifications have been enabled, additional Request Completion Event calls have no effect. The handler will be called once when the next entry is added to the CQ specified as a modifier to this Verb. The presence of Solicited Events may impact this behavior. See 11.4.2.2 Re-

quest Completion Notification on page 535 & 9.2.3 Solicited Event (SE) - 1 bit on page 209 for details.

C10-111: A CQ **shall** have at most one Completion Event notification request outstanding.

C10-112: A CI **shall** generate a single Completion Event when a Work Completion that satisfies the outstanding Completion Event request is added to the CQ.

C10-113: A CI **shall not** generate a Completion Event for existing Work Completion entries on the specified CQ at the time the completion notification request is registered.

A notification will not be generated until the next entry is added to the CQ.

The following sequence of calls should be used when using Request Completion Notification in order to ensure that a new CQ entry is not missed for the specified CQ.

1) Poll for Completion to dequeue existing CQ entries.
2) Request Completion Notification.
3) Poll for Completion to pick up any CQ entries that were added between the time the first Poll for Completion was called and the notification is enabled.

If a handler has not been registered, a notification will not be generated.

When the handler routine is invoked, an indication of which CQ has generated the completion notification will be supplied. Once the handler routine has been invoked, the Consumer must call Request Completion Notification again to be notified when a new entry is added to the CQ.

C10-114: For each Completion Event, the CI **shall** indicate which CQ caused the generation of that event.

The Consumer is responsible for polling the CQ to retrieve the work completion. This function is not performed automatically when the notification occurs.

10.9 PARTITIONING

This section discusses InfiniBand™ support for partitioning of an InfiniBand™ network. The Verb support for partitioning is contained in the Verbs that perform Queue Pair management, read Channel Interface (CI) content, and set it. These are documented in 11.2 Transport Resource Management on page 476.

In this discussion, the term "Partition Manager" (PM) refers to the function of the Subnet Manager that deals with partitioning for the CI being discussed; see 13.5 MAD Processing on page 630 for how SMPs are directed to that manager.

10.9.1 INTRODUCTION

Partitioning enforces isolation among systems sharing an InfiniBand™ fabric by requiring that packets contain a 16-bit Partition Key (P_Key) which must match a P_Key stored at the receiver or be discarded; see definition of "match" below (10.9.3 Partition Key Matching). There are no Verbs that directly set the P_Keys sent or matched against in a CA. Verbs instead specify an index into a table of P_Keys: the P_Key_ix, specifying an entry in the P_Key Table. The contents of the P_Key Table are controlled by the subnet's Partition Manager (PM), which sets them using Subnet Management Packets (SMPs) sent through the subnet's Subnet Manager.

Subsections appearing below describe the P_Key Table, the matching process, and the way P_Keys are attached to packets. See 14.2.5 Attributes on page 658 for a description of the SMPs which set entries in the P_Key Table.

10.9.1.1 LIMITED AND FULL MEMBERSHIP

A collection of endnodes with the same P_Key in their P_Key Tables are referred to as being *members of a partition*, or *in a partition*. A P_Key Table can specify one of two types of partition membership: Limited or Full. The high-order bit of the partition key is used to record the type of membership in a partition table: 0 for Limited, and 1 for Full. Limited members cannot accept information from other Limited members, but communication is allowed between every other combination of membership types.

10.9.1.2 SPECIAL P_KEYS

There are P_Keys that have special meaning: the default partition key, and the invalid partition keys.

C10-115: The P_Key value 0xFFFF **shall** represent the default partition key.

The default partition key provides Full membership in the default partition.

C10-116: The CI **shall** regard a P_Key as invalid if its low-order 15 bits are all zero.The CI **shall** mark a table entry as invalid by filling it with an invalid P_Key.

C10-117: The PM **must not** use these two P_Key values for any other purposes.

Any P_Key which is not invalid is referred to as valid. The default partition key is valid. A P_Key Table entry containing a valid P_Key is referred to as a valid P_Key Table entry.

10.9.1.3 OPERATION ACROSS SUBNETS

C10-118: Switches or Routers **shall not** modify P_Key values when packets are forwarded/routed within or between subnets.

C10-119: A packet's P_Key **must** match a P_Key stored at the destination CI or CA, or the packet **shall** be discarded; see the definition of "match" below (10.9.3 Partition Key Matching).

In the above case a P_Key sourced in one subnet must be valid in another subnet. Since subnets may have different PMs, this must be arranged to happen, for example by human administration (analogous to assignment of static IP addresses) or by a program dialog between subnets' PMs. The definition of the messages used in such an inter-PM dialog is beyond the scope of this version of the specification.

10.9.2 THE PARTITION KEY TABLE (P_KEY TABLE)

C10-120: Each HCA port and switch SMA port **shall** contain a Partition Key Table (P_Key Table). The valid entries in the P_Key Table **shall** hold P_Keys for all the endnodes with which this CI can communicate.

If a switch or router supports the optional P_Key Enforcement feature, then each of its ports shall contain a Partition Key Table (P_Key Table).

C10-121: The P_Key Table size, meaning the maximum number of entries it can hold, **must** be greater than or equal to one and less than or equal to 65535.

The maximum number of entries that can be held in a P_Key Table can be obtained by using the Query HCA Verb or the NodeInfo SMP. (See 11.2.1.2 Query HCA on page 476 and 14.2.5.3 NodeInfo on page 662.)

C10-122: The CI **must not** provide any interface which allows software above the Verbs to alter the P_Key Table contents or change the validity of any entry in the P_Key Table, except through the use of SMPs.

Verbs allow host software to read entries in the P_Key Table. If the value read is an invalid partition key value, that entry is invalid.

SMPs sent to the endnode are used to read and write entries in the P_Key Table. The operations involved when a table is written are described in a later section.

C10-123: If non-volatile storage is not used to hold P_Key Table contents, then if a PM (Partition Manager) is not present, and prior to PM initialization of the P_Key Table, the P_Key Table **must** act as if it contains a single valid entry, at P_Key_ix = 0, containing the default partition key. All other entries in the P_Key Table **must** be invalid.

10.9.3 PARTITION KEY MATCHING

C10-124: The P_Key field of incoming packets received by an endnode **shall** be matched against a resident P_Key as described in the remainder of this section.

Also see 9.6.1.1.3 BTH:P_Key on page 241 and 18.2.1 Attributes on page 846.

In the following, let M_P_Key (Message P_Key) be the P_Key in the incoming packet and E_P_Key (Endnode P_Key) be the P_Key it is being compared against in the packet's destination endnode.

- If:
  - neither M_P_Key nor E_P_Key are the invalid P_Key,
  - and the low-order 15 bits of the M_P_Key match the low order 15 bits of the E_P_Key;
  - and the high order bit (membership type) of both the M_P_Key and E_P_Key are not both 0 (i.e., both are not Limited members of the partition)

  then the P_Keys are said to *match*. In this case the incoming packet is accepted and processed normally.
- In all other cases the P_Keys are said to *not match*. The incoming packet must be treated as if it was sent to a nonexistent device, meaning:
  - no ACK is returned
  - optionally, a trap SMP is sent to the SM and a counter is incremented; see 10.9.4 Bad P_Key Trap and P_Key Violations Counter (Optional)
  - there is no other effect on the target endnode.

10.9.4 BAD P_KEY TRAP AND P_KEY VIOLATIONS COUNTER (OPTIONAL)

o10-55: If the CA ports and the GSI port for switches and routers support the trap SMP for P_Key Violations, then if a packet's P_Key does not match, the destination node **shall** send a trap SMP to the SM, specifying the partitioning class and the Bad P_Key Notification method. The body of the trap SMP **must** contain the header(s) of the offending packet. Like all traps, this one **shall not** be sent at a frequency faster than the Subnet Timeout.

o10-56: If the CA ports and the GSI port for switches and routers support the trap SMP for P_Key Violations, then if another P_Key mismatch occurs before the trap can be sent, the data for the new mismatch **shall** replace the previously stored data.

o10-57: If the CA ports and the GSI port for switches and routers support a P_Key Violations counter, then it **shall** have the following characteristics:

- Its minimum size is one bit; its maximum size is 16 bits (unsigned).
- It is incremented whenever the P_Key on a message arriving on a given port does not match (as described in 10.9.3 Partition Key Matching).
- When its value reaches all 1s, further incrementing does not change its value: i.e., it saturates.
- It is initialized by power on reset to zero.

The P_Key Violations counter can be read and set by using a SMP that accesses P_keyViolations component of the PortInfo attribute; see 14.2.5.1 Notices and Traps on page 660.

10.9.5 CI Partition Support

C10-125: Except for the subnet management QP (QP0) and QPs providing RD (Reliable Datagram) or Raw Datagram service, a P_Key **must** be associated with each QP before the QP is used. If a CI has multiple ports, the P_Key Table to which the P_Key index refers **shall** be the P_Key Table of the port that the QP is currently using.

This association is done through Verbs that specify the P_Key_ix of the key to use.

C10-126: The CI **shall** attach a QP's P_Key to all packets sent from the QP's send queue, except for SMPs, raw datagram packets and packets sent from RD QPs.

SMPs are always sent with the default P_Key, Raw datagram packets do not contain a P_Key, and packets from an RD QP get their P_Keys from the EE context associated with the RD QP.

C10-127: The CI **shall** compare the QP's P_Key to the P_Key contained in all incoming packets, except for raw packets and packets destined for QP0, QP1, and QPs providing RD service.

The comparison is described in 10.9.3 Partition Key Matching. The exceptions to this are described in 10.9.8 Partition Enforcement on Management Queue Pairs and 10.9.5.1 EE Context (Reliable Datagram) Support.

10.9.5.1 EE CONTEXT (RELIABLE DATAGRAM) SUPPORT

o10-58: If the CI supports the RD Service, then it **must** associate a P_Key with each EE Context before the EE Context is used. If a CI has multiple ports, the P_Key Table to which the P_Key index refers **shall** be the P_Key Table of the port that the EE Context is currently using.

o10-59: If the CI supports the RD Service, then the CI **must** attach an EE Context's P_Key to all outgoing Reliable Datagram (RD) packets emitted using that EE Context. All incoming packets using a given EE context **shall** be compared with that EE Context's P_Key as described in 10.9.3 Partition Key Matching.

As stated in that section: if the P_Keys match, the packet is processed normally; otherwise it is silently discarded and, optionally, a trap is issued and the Bad P_Key Counter is incremented as described in that section.

RD service is not used on management queue pairs, so this EE Context support does not apply to them.

10.9.5.2 PARTITION KEY CHANGES

C10-128: When the PM sends a message to a CI port requesting a change to the value of a P_Key Table element, the CI **must** return a response message indicating that the action has either been carried out successfully or not performed for some reason.

C10-129: The CI **shall** guarantee that, after the point in time when it sends a response message to the PM indicating success, the updated P_Key Table values will be used to process all subsequent incoming and outgoing packets traversing the associated port.

This behavior may have begun prior to the PM's receiving the success reply.

10.9.6 TCA PARTITION SUPPORT

C10-130: TCA support for partitioning **must** be the same as that for CIs, with the exception that association of a P_Key with a queue **shall** be done in response to messages that initiate creation of queue pairs, as part of establishing communication with another endnode.

In all other respects, the TCA behaves exactly like a CI in terms of multiple ports, incoming packets, outgoing packets, and changes to the P_Key Table.

10.9.7 FABRIC PARTITION SUPPORT

The switches in the InfiniBand™ fabric may optionally also enforce partitioning. How P_Keys are loaded into switches and how they are used is described in several sections of the chapter describing switches (18.2.4.2.1 Inbound P_Key Enforcement on page 850 and 18.2.4.4.1 Outbound P_Key Enforcement on page 858.

10.9.8 Partition Enforcement on Management Queue Pairs

The two types of management queues each treat partition enforcement in a different way.

C10-131: Packets sent to the Subnet Management Interface QP **shall** always be accepted, regardless of the P_Key contained in the packet.

Isolation and security of management communication are not provided by partitioning, but instead by checking of the Management Key.

Packets sent from a Subnet Management Interface QP may have any P_Key; the default P_Key is used by convention, as described in the management sections.

C10-132: Packets sent to the General Service Interface QP (QP1) **shall** be accepted if the P_Key in the packet matches any valid P_Key in the P_Key Table of the port on which the packet arrived. Matching is defined in 10.9.3 Partition Key Matching.

As stated in that section: if the P_Keys match, the packet is processed normally; otherwise it is silently discarded and, optionally, a trap is issued and the Bad P_Key Counter is incremented as described in that section.

C10-133: Packets sent from the Send Queue of a GSI QP **shall** attach a P_Key associated with that QP, just as a P_Key is associated with non-management QPs.

C10-134: Each switch **shall** also check P_Keys on its GSI QP. Switches **shall** support a P_Key table with at least one entry against which the P_Key of packets destined for the switch's GSI **shall** be matched, according to the rules as stated in C10-132: above.

10.9.9 Related Enforcement of Management Message Checking

Checking of the M_Key (see 14.2.4 Management Key on page 654) can optionally be used to prevent anything but an authorized subnet manager from reading any SM data from the SMI, and when the protection test fails, silently discarding the packet that failed. Similarly preventing the writing of SM data through the SMI, with silent discard, is mandatory.

In addition, it is an option to store the M_Key(s), the M_KeyProtectBits which control M_Key checking, and the lease period across power cycles Table 126 PortInfo on page 665.Thus, for example, system initialization techniques cannot assume that a constant default value for that data is present except for first-power-on from the factory.

10.10 Error Handling Semantics and Mechanisms

This section describes the types of errors that are detected at the Channel interface and the response that is generated when those error events occur.

10.10.1 Error Types

Three classes of errors reported through the Verbs have been defined: immediate errors, completion errors and asynchronous errors. Each of these error classes are described in more detail under their respective headings within 10.10.2 Error Handling Mechanisms. A brief description of each error class follows.

Immediate errors are returned as status from the Verbs.

Completion errors are returned to the Verbs Consumer as status within a Work Completion.

Asynchronous errors are returned through an event handling mechanism.

10.10.2 Error Handling Mechanisms

This section describes the mechanisms used to notify the Verb Consumer of errors in the requested operations.

10.10.2.1 Immediate Errors

C10-135: The CI **shall** return Immediate errors upon return of control from the Verb to the Consumer.

The details of these error types are included with each Verb described in the Verbs chapter.

C10-136: If an immediate error is returned from a Verb involved in posting Work Requests to a queue, the CI **shall** ensure that the Work Request has not been posted to the queue.

10.10.2.2 Completion Errors

C10-137: A Work Request or WQE that is “completed in error” **shall** have the appropriate completion error returned in the Work Completion status.

The complete list of errors that can be returned in the Work Completion status is described in the Verbs chapter under the Completion Queue Operations (11.4.2.1 Poll for Completion on page 531).

There are two classes of completion errors: Interface checks and processing errors. An interface check is an error in the information supplied to the Channel Interface detected before data is placed onto the link. A processing error is an error encountered during the processing of the work request by the Channel Interface.

10.10.2.3 ASYNCHRONOUS ERRORS

Consumers are notified about asynchronous errors through an asynchronous notification mechanism. In order to be notified when asynchronous errors occur, the Consumer must register a handler using the Set Asynchronous Event Handler Verb.

C10-138: After the asynchronous event handler is registered, all subsequent asynchronous errors **shall** result in a call to the error handler. Asynchronous errors that occur before the error handler is registered **shall** be lost.

The details of these errors are discussed in 11.6.3.2 Affiliated Asynchronous Errors on page 543 and 11.6.3.3 Unaffiliated Asynchronous Errors on page 544

C10-139: Only one error handler **shall** be registered per HCA. Subsequent calls to the Set Asynchronous Error Handler Verb **shall** cause the previous handler address to be overwritten with the new handler address.

There are two Asynchronous error types:

- Unaffiliated Asynchronous Error. Not related to any specific WQ or CQ.

C10-140: Unaffiliated Asynchronous Errors are handled according to type: local catastrophic errors **shall** place all QP/EEs in the Error State; local port errors **shall** have no effect on QP/EE State.

- Affiliated Asynchronous Error. Related to a specific WQ, CQ or EE context and unable to report the error in a completion. The QP or EE context is transitioned to the Error State.

10.10.3 EFFECTS OF ERRORS ON QP SERVICE TYPES

The different types of IB errors defined have varying effects on queue processing dependent upon the QP's Service Type.

It is important to note that catastrophic errors on the local QP have no direct effect on the remote QP. No attempt is made to send a message below the Verbs to tear down a connection just because a QP has encountered an error. However, NAK codes which are generated as the result of a QP being in the error state will have an effect on the QP receiving those NAK codes.

10.10.3.1 RELIABLE CONNECTION QPS:

C10-141: Immediate errors **shall not** affect QP processing since the Work Request never gets posted to the QP.

C10-142: For Send Queue completion errors, the Work Request on the Send Queue in which the error occurred **shall** be completed in error by the CI. The QP **shall be** placed in the Error State. All subsequent Work Requests **shall** be completed in error

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. If the local error was an interface check, the remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error. If the local error was a processing error, the remote, corresponding Receive Queue may or may not complete a Work Request in error. The condition of the local and remote memory when a completion error occurs on the send queue for RDMA and atomic operations is specified in 10.3.1.7 Error.

C10-143: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP **shall be** placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-144: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** be transitioned to the Error State. Any request in progress on the corresponding Work Queue **shall** be halted and **shall not** be completed successfully.

C10-145: Table 68 Completion Error Handling for RC Send Queues and Table 69 Completion Error Handling for RC Receive Queues are a more detailed description of the RC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 540

.

Table 68 Completion Error Handling for RC Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Length	Interface	Error	None
Local Operation	Interface	Error	None

Table 68 Completion Error Handling for RC Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Operation	Processing	Error	None
Local Protection	Interface	Error	None
Local Protection	Processing	Error	None
Memory Window Bind	Interface	Error	None
Invalid Request	Processing	Error	Error
Remote Access	Processing	Error	None
Remote Operation	Processing	Error	Error
RNR NAK Retry Counter Exceeded	Processing	Error	None
Transport Retry Counter Exceeded	Processing	Error	None

Table 69 Completion Error Handling for RC Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Length	Processing	Error	Error when NAK received
Local Protection	Processing	Error	Error when NAK received
Local Operation	Processing	Error	Error when NAK received

10.10.3.2 RELIABLE DATAGRAM QPS:

o10-60: If the CI supports RD Service, immediate errors **shall** have no effect on QP/EE processing since the Work Request never gets posted to the QP/EE.

o10-61: If the CI supports RD Service, completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. If the local error was an interface check, the remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error. If the local error was a processing error, the remote, corresponding Receive Queue may or may not complete a Work Request in error. The condition of the local and remote memory when a completion error occurs on the send queue for RDMA and atomic operations is specified in 10.3.1.6 Send Queue Error (SQEr).

o10-62: If the CI supports RD Service, for local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. All subsequent Work Requests **shall not** be affected by the error.

o10-63: If the CI supports RD Service, completion errors **shall** have no effect on the EE Context State.

o10-64: If the CI supports RD Service, Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall** be halted and **shall not** be completed successfully.

o10-65: If the CI supports RD Service, when an Affiliated Asynchronous Error is associated only with the QP, the error **shall** have no effect on the EE context. If an Affiliated Error is associated with the EE context, the EE context **shall** transition to the Error state.

o10-66: If the CI supports RD Service, Table 70 Completion Error Handling for RD Send Queues and Table 71 Completion Error Handling for RD Receive Queues are a more detailed description of the RD error handling actions that **must** be supported by the CI for RD:

Table 70 Completion Error Handling for RD Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State	Effect on Local EE State	Effect on Remote EE State	Error Handling Action
Local Length	Interface	SQ Error	None	None	None	1
Local Operation - QP	Interface	SQ Error	None	None	None	1
Local Operation - QP	Processing	SQ Error	Rcv WC Err	None	None	1, 3, 6
Local Operation - EE	Processing	SQ Error	Indeterminate	EE Error	None	1, 5
Local Protection	Interface	SQ Error	None	None	None	1
Local Protection	Processing	SQ Error	Rcv WC Err	None	None	1, 3, 6
Remote Operation - QP	Processing	SQ Error	Error	None	None	1
Remote Operation - EE	Processing	SQ Error	Error	Error	Error	1, 3, 5
Memory Window Bind	Interface	SQ Error	None	None	None	1

Table 70 Completion Error Handling for RD Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State	Effect on Local EE State	Effect on Remote EE State	Error Handling Action
Remote Access	Processing	SQ Error	None	None	None	1
Remote Operation - QP	Processing	SQ Error	Error	None	None	1, 3
Remote Invalid Request	Processing	SQ Error	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	None	None	1, 3
Local RDD Violation	Processing	SQ Error	None	None	None	1
Remote Invalid RD Request	Processing	SQ Error	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	None	None	1, 3
Transport Timeout Retry Counter Exceeded	Processing	SQ Error	None	Error	None	1
RNR NAK Retry Counter Exceeded	Processing	SQ Error	None	None	None	1

Table 71 Completion Error Handling for RD Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state	Effect on local EE state	Effect on remote EE state	Error Handling Action
Local Length	Processing	Rcv WC Err	SQ Error	None	None	1, 3
Local Operation - QP	Processing	Rcv WC Err	SQ Error	None	None	1, 3
Local Operation - EE	Processing	Rcv WC Err	SQ Error	EE Error	EE Error	1, 3, 5
Local Protection	Processing	Rcv WC Err	SQ Error	None	None	1, 3
Remote Invalid Request	Processing	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	SQ Error	None	None	1, 3
Remote Invalid RD Request	Processing	None if 1st packet. Opt Rcv WC Err if other than 1st packet.	QP Error	None	None	2

Table 71 Completion Error Handling for RD Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state	Effect on local EE state	Effect on remote EE state	Error Handling Action
Remote aborted	Processing	Rcv WC Err	Indeterminate[a]	None	None	3, 1[b]

a. May be in SQError or may retry the message later using a different RQ WQE.
b. Action 1 will only happen in the case where the requester abandoned the operation.

Error Handling Actions:

Uninvolved SQs and RQs are unaffected unless they attempt to use an EE-context or QP that is in the error state.

1) The SQ active over the EE-context at the time the error occurred goes to the SQEr state.
   - Receives for the RQ associated with the local SQ placed in SQEr state continue as normal (i.e. are not completed in error, unless they also experience a separate error).
   - Remainder of the WQEs in the SQ which experienced the error are returned in error via Work Completions (WCs).
2) The SQ active over the EE-context at the time the error occurred causes the full QP (associated with the SQ) to be placed in the error state.
   - Remainder of sends for the SQ that caused the error are returned in error via WCs.
   - Any receives associated with the local RQ are returned in error via WCs.
3) RQ active over the EE-context at the time the error occurred has the WQE which experienced the error returned in error via a WC.
   - All other WQEs in that RQ continue as normal (i.e. are not completed in error, unless they also experience an error).
   - Sends for the SQ associated with the local RQ placed in SQEr state continue as normal (i.e. are not completed in error, unless they also experience a separate error).
4) RQ active over the EE-context at the time the error occurred causes the full QP (associated with the RQ) to be placed in the error state.
   - Remainder of receives for the RQ that caused the error are returned in error via WCs.
   - Any sends associated with the local SQ are returned in error via WCs.

5) Local EE-context is placed in the error state. Remote EE-context state is indeterminate (i.e. may not be in the error state, for example as a result of a source timeout).
   - EE-context cannot be resumed.
   - Must re-establish EE-context.

6) When a Local Protection or Operation SQ Error occurs on RD QPs, the CI on the local side shall emit an Infiniband no-op (RDMA Write of length 0 with no immediate data) below the Verbs to the RD RQ associated with the local error, assuming the RD channel is still operational. This will cause the in-process RQ Work Request on the remote side to be completed in error. The Receive side cannot depend on receiving that message.

10.10.3.3 UNRELIABLE CONNECTED QPS:

C10-146: Immediate errors **shall** have no effect on QP processing since the Work Request never gets posted to the QP.

C10-147: Completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. The remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error. The condition of the remote memory when a completion error occurs on the send queue for RDMA Write operations is specified in 10.3.1.6 Send Queue Error (SQEr).

C10-148: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP is placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-149: Table 72 Completion Error Handling for UC Send Queues and Table 73 Completion Error Handling for UC Receive Queues are a more detailed description of the UC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 540.

Table 72 Completion Error Handling for UC Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Length	Interface	SQ Error	None
Local Operation	Interface	SQ Error	None
Local Operation	Processing	SQ Error	None
Local Protection	Interface	SQ Error	None
Local Protection	Processing	SQ Error	None
Memory Window Bind	Interface	SQ Error	None
Invalid Request	Processing	SQ Error	Error

Table 73 Completion Error Handling for UC Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Length	Processing	Error	None
Local Protection	Processing	Error	None
Local Operation	Processing	Error	None

C10-150: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall** be halted and **shall not** be completed successfully.

10.10.3.4 UNRELIABLE DATAGRAM QPS:

C10-151: Immediate errors **shall** have no effect on QP processing since the Work Request never gets posted to the QP.

C10-152: Completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. The remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error.

C10-153: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP is placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-154: Table 74 Completion Error Handling for UD Send Queues and Table 75 Completion Error Handling for UD Receive Queues provide a more detailed description of the UC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 540.

Table 74 Completion Error Handling for UD Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Length	Interface	SQ Error	None
Local Operation	Interface	SQ Error	None
Local Operation	Processing	SQ Error	None
Local Protection	Interface	SQ Error	None
Local Protection	Processing	SQ Error	None
Invalid Request	Processing	SQ Error	Error

Table 75 Completion Error Handling for UD Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Length	Processing	Error	None
Local Protection	Processing	Error	None
Local Operation	Processing	Error	None

C10-155: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall** be halted and **shall not** be completed successfully.

10.10.3.5 RAW QPS:

C10-156: Immediate errors **shall** have no effect on QP processing since the Work Request never gets posted to the QP.

C10-157: Completion errors on a Send Queue **shall** result in Send Queue processing being halted and the Send Queue state **shall** transition to the Send Queue Error State, as per the state diagram. The Work Request where the error occurred **shall** be completed in error.

In the case of local send queue errors, any and all Work Requests on the Send Queue in which the error occurred are completed in error by the Channel Interface. The remote, corresponding Receive Queue will not consume a Work Request and thus will not surface a completion error.

C10-158: For local Receive Queue completion errors, the Work Request on the Receive Queue in which the error occurred **shall** be completed in error by the CI. The QP is placed in the Error State. All subsequent Work Requests **shall** be completed in error.

C10-159: Table 76 Completion Error Handling for Raw Datagram Send Queues and Table 77 Completion Error Handling for Raw Datagram Receive Queues provide a more detailed description of the UC error handling actions that **must** be supported by the CI according to the error and Work Queue type.

Descriptions of the error types used in the table are contained in 11.6.2 Completion Return Status on page 540.

Table 76 Completion Error Handling for Raw Datagram Send Queues

Error Type	Completion Type	Effect on Local QP State	Effect on Remote QP State
Local Length	Interface	SQ Error	None
Local Operation	Interface	SQ Error	None
Local Operation	Processing	SQ Error	None
Local Protection	Interface	SQ Error	None
Local Protection	Processing	SQ Error	None
Invalid Request	Processing	SQ Error	Error

Table 77 Completion Error Handling for Raw Datagram Receive Queues

Error Type	Completion Type	Effect on local QP state	Effect on remote QP state
Local Length	Processing	Error	None
Local Protection	Processing	Error	None
Local Operation	Processing	Error	None

C10-160: Affiliated Asynchronous Errors **shall** result in the QP processing being halted such that outstanding Work Requests are not completed successfully by the Channel Interface. The QP **shall** transition to the Error State. Any request in progress on the corresponding Work Queue **shall** be halted and **shall not** be completed successfully.

CHAPTER 11: SOFTWARE TRANSPORT VERBS

11.1 VERBS INTRODUCTION AND OVERVIEW

The Verbs described in this chapter provide an abstract definition of the functionality provided to a host by a host channel interface. Host CIs which are compliant with this specification must exhibit the semantic behavior described by the Verbs.

Since the Verbs define the behavior of the host CI, they may influence the design of software constructs, such as application programming interfaces (APIs), which provide access to the host CI. However, this specification explicitly does not define any such API. In particular, there is no requirement that an API used with a compliant host CI be semantically consistent with the Verbs.

11.1.1 VERB CLASSES

11.1.1.1 MANDATORY VS. OPTIONAL VERBS

Some Verbs are mandatory, and some are required only if an optional feature is supported.

C11-1: A CI **shall** support all Verbs classified as mandatory in Verb Classes.

C11-2: If a CI claims conformance to an optional feature, the CI **shall** support all Verbs associated with that optional feature as indicated in Verb Classes.

11.1.1.2 MANDATORY VS. OPTIONAL VERB FUNCTIONALITY

Some Verbs define functionality that applies only if certain optional features are supported.

C11-3: If a CI supports a given Verb, the CI **shall** support all functionality defined for that Verb that's not indicated as being optional.

C11-4: If a CI supports a given Verb and claims conformance to an optional feature, the CI **shall** support all functionality defined for that Verb that's associated with that optional feature.

11.1.1.3 CONSUMER ACCESSIBILITY

Verb Consumers are the direct users of the Verbs, and are sub-divided into two classes, Privileged and User-Level.

Privileged Consumers are typically those Consumers that operate at a privilege level sufficient to access OS internal data structures directly, and that have the responsibility to control access to the Channel Interface. All Verbs are available for use by Privileged Consumers.

User-Level Consumers are those Consumers that must rely on another agent, having a sufficient high level of privilege, to manipulate OS data structures. Only those Verbs specifically labeled as such are available for use by User-Level Consumers

Table 78 Verb Classes

Verb	Mandatory/Optional Classification	Consumer Accessibility
Open HCA	Mandatory	Privileged
Query HCA	Mandatory	Privileged
Modify HCA Attributes	Access violation counters	Privileged
Close HCA	Mandatory	Privileged
Allocate Protection Domain	Mandatory	Privileged
Deallocate Protection Domain	Mandatory	Privileged
Allocate Reliable Datagram Domain	RD Service	Privileged
Deallocate Reliable Datagram Domain	RD Service	Privileged
Create Address Handle	Mandatory	User-Level and Privileged
Modify Address Handle	Mandatory	User-Level and Privileged
Query Address Handle	Mandatory	User-Level and Privileged
Destroy Address Handle	Mandatory	User-Level and Privileged
Create Queue Pair	Mandatory	Privileged
Modify Queue Pair	Mandatory	Privileged
Query Queue Pair	Mandatory	Privileged
Destroy Queue Pair	Mandatory	Privileged
Get Special QP	Mandatory	Privileged
Create Completion Queue	Mandatory	Privileged
Query Completion Queue	Mandatory	Privileged
Resize Completion Queue	Mandatory	Privileged

Table 78 Verb Classes

Verb	Mandatory/Optional Classification	Consumer Accessibility
Destroy Completion Queue	Mandatory	Privileged
Create EE Context	RD Service	Privileged
Modify EE Context Attributes	RD Service	Privileged
Query EE Context	RD Service	Privileged
Destroy EE Context	RD Service	Privileged
Register Memory Region	Mandatory	Privileged
Register Physical Memory Region	Mandatory	Privileged
Query Memory Region	Mandatory	Privileged
Deregister Memory Region	Mandatory	Privileged
Reregister Memory Region	Mandatory	Privileged
Reregister Physical Memory Region	Mandatory	Privileged
Register Shared Memory Region	Mandatory	Privileged
Allocate Memory Window	Mandatory	Privileged
Query Memory Window	Mandatory	Privileged
Bind Memory Window	Mandatory	User-Level and Privileged
Deallocate Memory Window	Mandatory	Privileged
Attach QP to Multicast Group	UD Multicast Service	Privileged
Detach QP from Multicast Group	UD Multicast Service	Privileged
Post Send Request	Mandatory	User-Level and Privileged
Post Receive Request	Mandatory	User-Level and Privileged
Poll for Completion	Mandatory	User-Level and Privileged
Request Completion Notification	Mandatory	User-Level and Privileged
Set Completion Event Handler	Mandatory	Privileged
Set Asynchronous Event Handler	Mandatory	Privileged

11.2 TRANSPORT RESOURCE MANAGEMENT

11.2.1 HCA

11.2.1.1 OPEN HCA

Description:

Opens the specified HCA and returns an opaque object or handle to uniquely reference each HCA so that Consumers can distinguish between HCAs in the endnode.

C11-5: The handles returned for different HCAs within a system **shall** all be unique.

Once opened, a specific HCA cannot be opened again until after it is closed. Opening the HCA prepares the HCA for use by the Consumer.

C11-6: If Open HCA is called for an HCA that is currently open, the CI shall return the HCA already in use error.

Input Modifiers:

- The unique identifier for this HCA. The naming scheme is defined by the OSV.

Output Modifiers:

- A handle for the HCA instance used as a modifier to other Verbs to specify the desired target HCA.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA name.
  - HCA already in use.

11.2.1.2 QUERY HCA

Description:

Returns the attributes for the specified HCA.

The maximum values defined in this section are guaranteed not-to-exceed values. It is possible for an implementation to allocate some HCA resources from the same space. In that case, the maximum values returned are not guaranteed for all of those resources simultaneously.

Input Modifiers:

- HCA handle.

Output Modifiers:

- The HCA attributes returned are:
  - Vendor specific information such as:
    - Vendor ID.
    - Vendor supplied Part ID.
    - Hardware version.

HCA specific values:

- The maximum number of QPs supported by this HCA.
- The maximum number of outstanding work requests on any Work Queue supported by this HCA.
- The maximum number of scatter/gather entries per Work Request supported by this HCA, for all Work Requests other than Reliable Datagram Receive Queue Work Requests.
- The maximum number of scatter/gather entries per Reliable Datagram Receive Queue Work Request supported by this HCA. Zero if RD Service is not supported.
- The maximum number of CQs supported by this HCA.
- The maximum number of entries in each CQ supported by this HCA.
- The maximum number of Memory Regions supported by this HCA.
- The largest contiguous block that can be registered by this HCA, specified in bytes.
- The maximum number of Protection Domains supported by this HCA.
- The memory page sizes supported by this HCA.
- Number of physical ports on this HCA.
- Port Attribute list (one list for each port on this HCA):
  - MTU and message size supported for each port of this HCA.
  - Base LID & LMC for each port of this HCA. These values are valid only when the Port State of the port is Armed or Active. For other port states the values returned are indeterminate. (For more information on the port state see 14.4.4 Port State Transitions on page 697).
  - Contents and length of the Source GID Table. The value of Assigned GIDs are valid only when Port State is Armed or Active. For other states the value of assigned GIDs is indeterminate.

- PortState of each port of this HCA. see 7.2.7 State Machine Terms on page 143.
- Contents and length of the partition table. A partition table is required per port. The contents of the partition table are valid only when the Port State is Armed or Active. For other states the contents of the partition table are implementation dependent.
- The maximum number of virtual lanes supported by this HCA.
- Optional Bad P_Key counter for each port supported by the HCA.
- Q_Key Violation counter for each port supported by the HCA.
- Contents of the Subnet Manager address information for each port of this HCA. This is a table, with entries arranged on a per HCA port basis, which contains the LID and Service Level of the Subnet Manager for that port. If this has not been set by the Subnet Manager (Port State is Armed or Active), this should be set to the permissive LID (0xFFFF).
- The following CapabilityMask bits for each port on this HCA as defined in the PortInfo CapabilityMask:
    - IsSM.
    - IsSMDisabled.
    - IsSNMPTunnelingSupported.
    - IsDeviceManagementSupported.
    - IsVendorClassSupported.

- Maximum number of partitions supported by this HCA. The number of partitions supported must be at least one.
- Node GUID for this HCA.
- The Local CA ACK Delay. This value specifies the maximum expected time interval between the local CA receiving a message and it transmitting the associated ACK or NAK. This is suggested for use in computing the “Local ACK Timeout” field in a CM REQ message, or the “Target ACK Delay” field in a CM REP message. See Local ACK Timeout and Target ACK Delay in Message Field Details. The delay value in microseconds is computed using 4.096μs * $2^{(\text{Local CA ACK Delay})}$. The delay value is not a guaranteed upper bound for the CA’s response time, but rather one that can be used as a “maximum expected value” for timeouts.
- Bad P_Key counter support indicator.

- Q_Key Violation counter support indicator.
- The maximum number of RDMA Reads & atomic operations that can be outstanding per QP with this HCA as the target. Shall apply to atomics only if this HCA supports atomic operations.
- The maximum number of RDMA Reads & atomic operations that can be outstanding per EE with this HCA as the target. Shall apply to atomics only if this HCA supports RD & atomic operations. For this version of the specification, this value is one.
- The maximum number of resources used for RDMA Reads & atomic operations by this HCA with this HCA as the target. Shall apply to atomics only if this HCA supports atomic operations.
- The maximum depth per QP for initiation of RDMA Read & atomic operations by this HCA. Shall apply to atomics only if this HCA supports atomic operations.
- The maximum depth per EE for initiation of RDMA Read & atomic operations by this HCA. Shall apply to atomics only if this HCA supports RD & atomic operations. For this version of the specification, this value is one.
- Ability of this HCA to support atomic operations as well as serialization of atomic operations between itself and other system components such as processors and other HCAs. Three levels of atomicity are defined for this version of the specification:
  - Atomic operations not supported.
  - Atomicity is guaranteed only between QPs on this HCA only.
  - Atomicity is guaranteed between this HCA and any other component, such as CPUs and other HCAs.
- The maximum number of EE contexts that can be supported by this HCA. Shall be zero if the HCA does not support Reliable Datagrams.
- Maximum number of RDDs supported by this HCA. The number of RDDs supported must be at least two. Shall be zero if the HCA does not support Reliable Datagrams.
- The maximum number of Memory Windows supported by this HCA.
- The maximum number of Raw IPv6 Datagram QPs supported by this HCA. Shall be zero if Raw IPv6 Datagrams are not supported.

- The maximum number of Raw Ethertype Datagram QPs supported by this HCA. Shall be zero if Raw Ethertype Datagrams are not supported.
- Ability of this HCA to support modifying the maximum number of outstanding Work Requests per QP.
- Maximum number of multicast groups supported by this HCA. Shall be zero if this HCA does not support IBA unreliable multicast.
- Maximum number of QPs which can be attached to multicast groups for this HCA. Shall be zero if this HCA does not support IBA unreliable multicast.
- Maximum number of QPs per multicast group supported by this HCA. Shall be zero if this HCA does not support IBA unreliable multicast.
- Ability of this HCA to support raw packet multicast.
- Ability of this HCA to support automatic path migration.
- Maximum number of Address Handles supported by this HCA.
- Ability of this HCA to change the primary physical port for a QP or EE when transitioning from SQD to RTS state.

- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete this request.
  - Invalid HCA handle.

11.2.1.3 MODIFY HCA ATTRIBUTES

Description:

Modifies the optional key counters in the HCA. Shall apply only if the HCA supports the Bad P_Key counter or the Invalid Q_Key counter.

Input Modifiers:

- HCA handle.
- Port Attribute list (one list for each port on this HCA):
  - Optional Bad P_Key counter for each port supported by the HCA.
  - Q_Key Violation counter for each port supported by the HCA.
  - The following CapabilityMask bits for each port on this HCA as defined in the PortInfo CapabilityMask:
    - IsSM.
    - IsSNMPTunnelingSupported.

- IsDeviceManagementSupported.
- IsVendorClassSupported.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid Counter specified.
  - Invalid Counter value.

11.2.1.4 CLOSE HCA

Description:

Closes and resets the specified HCA. This Verb is responsible only for deallocating resources allocated by the Channel Interface to prepare the HCA for use by the Consumer. All other resources are no longer associated or connected with the CI and are the responsibility of the Consumer to handle as deemed necessary.

Input Modifiers:

- HCA handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.

11.2.1.5 ALLOCATE PROTECTION DOMAIN

Description:

Allocates an unused Protection Domain object. Protection Domain objects are required when creating a Queue Pair or Address Handle, registering memory and allocating memory windows. A Protection Domain object provides an association between Queue Pairs, Address Handles, Memory Regions and Memory Windows. Operations on a Queue Pair that cause access to a Memory Region or a Memory Window are allowed only when the Protection Domain of the Queue Pair and the Protection Domain of the Memory Region or a Memory Window are identical.

Operations on an unreliable datagram queue pair are allowed only when the Protection Domain of the Queue Pair and the Protection Domain of the Address Handle contained in the work request are identical.

Input Modifiers:

- HCA Handle.

Output Modifiers:

- Protection Domain Object.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.

11.2.1.6 DEALLOCATE PROTECTION DOMAIN

Description:

Returns a previously Allocated Protection Domain object for reuse by the Allocate Protection Domain Verb. The Protection Domain object cannot be deallocated if it is still associated with any Queue Pair, Memory Region or Memory Window, or Address Handle.

Input Modifiers:

- HCA Handle.
- Protection Domain object.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid Protection Domain.
  - Protection Domain is in use.
  - Invalid HCA handle.

11.2.1.7 ALLOCATE RELIABLE DATAGRAM DOMAIN

Description:

Allocates an unused Reliable datagram domain object. Reliable datagram domain objects are required when setting up a reliable datagram Queue Pair and EE contexts. A reliable datagram domain object provides an association between Queue Pairs and EE contexts. Operations on a reliable datagram queue pair directed at an EE context are allowed only when the reliable datagram domain of the queue pair and the reliable datagram domain of the EE context are identical.

Input Modifiers:

- HCA Handle.

Output Modifiers:

- Reliable datagram domain object.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Reliable Datagrams not supported.

11.2.1.8 DEALLOCATE RELIABLE DATAGRAM DOMAIN

Description:

Returns a previously allocated reliable datagram domain object for reuse by the Allocate Reliable Datagram Domain Verb. The reliable datagram domain object cannot be deallocated if it is still associated with a Queue Pair or an EE context.

Input Modifiers:

- HCA Handle.
- Reliable datagram domain object.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid reliable datagram domain.
  - Reliable datagram domain is in use.
  - Invalid HCA handle.
  - Reliable Datagrams not supported.

11.2.2 ADDRESS MANAGEMENT VERBS

These Verbs create, manipulate and destroy address handles. These address handles are only used for Work Requests submitted to Unreliable Datagram Service Type QPs.

11.2.2.1 CREATE ADDRESS HANDLE

Description:

The purpose of the Create Address Handle Verb is to create an address handle for the address vector passed in through the Verbs. The normal completion for this Verb returns the address handle. The address handle is used to reference a local or global destination in all UD QP Post Sends.

Input Modifiers:

- HCA Handle.
- Protection domain
- Address vector, for UD transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - For global destination:
    - Flow label.
    - Hop limit.
    - Traffic class.
    - Source GID index.
  - For global destination or Multicast address:
    - Destination's GID (a.k.a. IPv6) address.
  - Maximum Static Rate.
  - Source Path Bits.

Output Modifiers:

- Address Handle.
- Verb results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid protection domain
  - Insufficient resources to complete request.

11.2.2.2 MODIFY ADDRESS HANDLE

Description:

The purpose of the Modify Address Handle Verb is to change an address vector associated with the address handle passed in by the Consumer.

Input Modifiers:

- HCA Handle.
- Address Handle.
- Address vector, for UD transports only, containing:
  - Service level.

- Send Global Routing Header Flag.
- Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
- For global destination:
    - Flow label.
    - Hop limit.
    - Traffic class.
    - Source GID index.
- For global destination or Multicast address:
    - Destination's GID (a.k.a. IPv6) address.
- Maximum Static Rate.
- Source Path Bits.

Output Modifiers:

- Verb results:
    - Operation completed successfully.
    - Invalid HCA handle.
    - Invalid address handle.

11.2.2.3 QUERY ADDRESS HANDLE

Description:

The purpose of the Query Address Handle Verb is to obtain the address vector associated with the address handle passed in by the Consumer.

Input Modifiers:

- HCA Handle.
- Address Handle.

Output Modifiers:

- Address vector, for UD transports only, containing:
    - Service level.
    - Send Global Routing Header Flag.
    - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
    - For global destination:
        - Flow label.
        - Hop limit.

        - Traffic class.
        - Source GID index.
    - For global destination or Multicast address:
        - Destination's GID (a.k.a. IPv6) address.
    - Maximum Static Rate.
    - Source Path Bits.
- Protection domain
- Verb results:
    - Operation completed successfully.
    - Invalid HCA handle.
    - Invalid address handle.

11.2.2.4 DESTROY ADDRESS HANDLE

Description:

The purpose of the Destroy Address Handle Verb is to remove an address vector and its associated address handle from the CI. After the address handle is removed, it can no longer be used to reference the destination.

Input Modifiers:

- HCA Handle.
- Address Handle.

Output Modifiers:

- Verb results:
    - Operation completed successfully.
    - Invalid HCA handle.
    - Invalid address handle.

11.2.3 QUEUE PAIR

11.2.3.1 CREATE QUEUE PAIR

Description:

Creates a QP for the specified HCA.

A set of initial QP attributes must be specified by the Consumer.

C11-7: If any of the required initial attributes are illegal or missing, an error **shall** be returned and the Queue Pair **shall not** be created.

On success, a handle to the newly created QP and the QP number are returned.

Input Modifiers:

- HCA handle.
- The QP attributes that must be specified at QP create time are:
  - The CQ to be associated with the Send Queue.
  - The CQ to be associated with the Receive Queue.
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Send Queue.
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue.
  - The maximum number of scatter/gather elements the Consumer will specify in a Work Request submitted to the Send Queue.
  - The maximum number of scatter/gather elements the Consumer will specify in a Work Request submitted to the Receive Queue.
  - Reliable datagram domain to be associated with this QP. Applicable only to RD QPs.
  - The Signaling Type must be specified for the Send Queue on this QP. The valid types are:
    - All Work Requests submitted to the Send Queue always generate a completion entry.
    - Consumer must specify on each Work Request submitted to the Send Queue whether to generate a completion entry for successful completions.
  - The Consumer must specify a Protection Domain.
  - The Transport Service Type requested for this QP. Valid Service Types are:
    - Reliable Connection.
    - Reliable Datagram.
    - Unreliable Connection.
    - Unreliable Datagram.

Output Modifiers:

- The handle for the newly created QP.
- QP number.
- The actual number of outstanding Work Requests supported on the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)

- The actual number of outstanding Work Requests supported on the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - Maximum number of Work Requests requested exceeds HCA capability.
  - Maximum number of scatter/gather elements requested exceeds HCA capability.
  - Invalid Protection Domain.
  - Invalid Service Type for this QP.
  - Invalid Reliable Datagram Domain.

11.2.3.2 MODIFY QUEUE PAIR

Description:

C11-8: Upon invocation of this Verb, the CI **shall** modify the attributes for the specified QP and then **shall** cause the QP to transition to the specified QP state.

Only a subset of the QP attributes can be modified in each of the QP states.

C11-9: If any of the QP attributes to be modified are invalid or the requested state transition is invalid, none of the QP attributes **shall** be modified. An immediate error **shall** be returned and the QP state **shall** remain unchanged.

Some QP attributes can be modified with outstanding Work Requests. WRs can be outstanding on the Receive Queue when the QP is in the Init, RTR, RTS & SQEr state and on the Send Queue when the QP is in the RTS state. Any outstanding Work Request on a Work Queue may not execute as expected if the QP modifiers are changed. For instance, if RDMA Reads, which were successfully posted, are outstanding when the QP is modified to no longer allow RDMA Reads, some outstanding in-flight RDMA Reads may complete while pending WRs may fail.

C11-10: The properties and requirements of the QP state transitions **shall** be supported as shown in Table 79.

Table 79 QP State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
Reset to Init	Enable/disable RDMA[a] and Atomic Operations. P_Key index. Physical port. Q_Key for unconnected Service Types.	None.	Enable posting to the Receive Queue.
Init to RTR	Remote Node Address Vector (Connected QPs only). Destination QP Number (RC/UC QPs only). RQ PSN. Number of responder resources for RDMA Read/atomic ops.	Alternate path address information (RC/UC QPs only). Enable/disable RDMA[a] and Atomic Operations. P_Key index. Q_Key. Number of WQEs. Minimum RNR NAK Timer Field (RC and RD QPs only).	Activate receive processing.
RTR to RTS	Local ACK Timeout (RC QP only) Retry count (RC QP only) RNR retry count (RC QP only) SQ PSN. Number of Outstanding RDMA Read/atomic ops at destination.	Enable/disable RDMA[a] & Atomic Operations. Q_Key. Alternate path address information (RC/UC QPs only). Path migration state. Number of WQEs. Minimum RNR NAK Timer Field (RC and RD QPs only).	Activate send processing.

Table 79 QP State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
RTS to RTS (no transition)	None.	Enable/disable RDMA Operations.[a] Q_Key. Alternate path address information (RC/UC QPs only). Path Migration state. Number of WQEs. Minimum RNR NAK Timer Field (RC and RD QPs only).	No transition.
SQEr to RTS	None.	Enable/disable RDMA & Atomic Operations.[a] Q_Key. Number of WQEs. Minimum RNR NAK Timer Field (RC and RD QPs only).	Activate send processing.
RTS to SQD	None.	None.	Deactivate send processing.

Table 79 QP State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
SQD to RTS	None.	Remote Node Address Vector (Connected QPs only). Alternate path address information (RC/UC QPs only). Channel migration state. Number of Outstanding RDMA Read/atomic ops at destination. Number of local RDMA Read/atomic responder resources. Q_Key. P_Key index. Timeout/Retry Information. Number of WQEs. Minimum RNR NAK Timer Field (RC and RD QPs only). Primary physical port associated with QP if HCA supports this capability (RC QPs only).	Activate send processing.
Any State to Error	None.	None allowed.	Queue processing is stopped. Work Requests pending or in process are completed in error, when possible.
Any state to Reset	None.	None allowed.	QP attributes are reset to the same values after the QP was created. Outstanding Work Requests are removed from the queues without notifying the Consumer.

a. If disable RDMA is requested while incoming RDMAs to that queue are in process, it is indeterminate when the disable will take effect. It is up to the Consumer to coordinate the disable with the remote QPs.

Input Modifiers:

- HCA handle.
- QP handle.
- The QP attributes to modify and their new values. The QP attributes that can be modified after the QP has been created are:
  - Next QP state. If specify the current state, only the QP attributes will be modified.
    - Enable or disable Send Queue Drained, Asynchronous Affiliated Event Notification. This modifier is only applicable when the next QP state chosen is SQD.
  - Primary P_Key index. Not applicable on a Raw Datagram or Reliable Datagram QPs.
  - The Q_Key that incoming Datagram messages are checked against and possibly used as the outgoing Q_Key (based on the WR Q_Key). This applies only to UD & RD QPs.
  - PSN for Send QP. Applicable only for RC, UD & UC QPs.
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Send Queue, if resizing of the work queues is supported by the HCA.
  - The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue, if resizing of the work queues is supported by the HCA.

  The following attributes are not applicable if the QP specified is a Special QP: SMI QP (QP0), GSI QP (QP1), Raw IPv6 and Raw Ethertype.

  - Primary physical port associated with this QP. Not applicable on RD QPs. Applicable for the SQD to RTS transition on RC QPs, if supported by the HCA.
  - PSN for Receive QP. Applicable only for RC & UC QPs.
  - Enable or disable incoming RDMA Reads on this QP. Not applicable on Unreliable Service Type QPs.
  - Enable or disable incoming RDMA Writes on this QP. Not applicable on UD Service Type QPs.
  - Enable or disable incoming Atomic Operations on this QP. Not applicable on Unreliable Service Type QPs.
  - Destination QP number. Applicable only to RC & UC QPs.
  - Number of RDMA Reads & atomic operations outstanding at any time. Applicable only to RC QPs.
  - Number of responder resources for handling incoming RDMA Reads & atomic operations. This value may be rounded up to a supported value, not to exceed the maximum value allowable for QPs for this HCA. Applicable only to RC QPs.

- Minimum RNR NAK Timer Field Value. When a message arrives which is targeted at a local receive queue, and that receive queue has no receive work requests outstanding, the CI may respond to the initiator with an RNR NAK packet. This modifier is the minimum value which shall be sent in the Timer Field of such an RNR NAK packet; it does not affect RNR NAKs sent for other reasons. If the value specified is not one of the RNR NAK Timer Field values defined in Table 45 Encoding for RNR NAK Timer Field on page 297, the CI shall return an immediate error. Applicable only to RC and RD QPs.
- Address vector, for RC & UC transports only, containing:
    - Service level.
    - Send Global Routing Header Flag.
    - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
    - Path MTU.
    - Maximum Static Rate.
    - Local ACK Timeout. Applicable only to RC QPs.
    - Retry count. Applicable only to RC QPs.
    - RNR retry count. Applicable only to RC QPs.
    - Source Path Bits.
    - For global destination:
        - Traffic class.
        - Flow label.
        - Hop limit.
        - Source GID index.
        - Destination's GID (a.k.a. IPv6) address.

- Alternate path address information, applicable only for RC & UC QPs when this CI support automatic path migration. Note: the path MTU for the alternate path must be the same as for the primary path. The specifics are:
    - Alternate path P_Key index.
    - Alternate path Physical port.
    - Alternate path address vector, containing:
        - Service level.
        - Send Global Routing Header Flag.
        - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.

- Maximum Static Rate.
- Local ACK Timeout. Applicable only to RC QPs.
- Retry count. Applicable only to RC QPs.
- RNR retry count. Applicable only to RC QPs.
- Source Path Bits.

For global destination:

- Traffic class.
- Flow label.
- Hop limit.
- Source GID index.
- Destination's GID (a.k.a. IPv6 address).

- Path Migration state. Valid only if this HCA supports automatic path migration. Valid states to set are:
  - Migrated.
  - Rearm.

Output Modifiers:

- The actual number of outstanding Work Requests supported on the Send Queue, if resizing of the work queues is supported by the HCA. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of outstanding Work Requests supported on the Receive Queue, if resizing of the work queues is supported by the HCA. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid QP handle.
  - Cannot change QP attribute.
  - Atomic operations not supported.
  - P_Key index out of range.
  - P_Key index specifies invalid entry in P_Key table.
  - Invalid QP state.
  - Invalid path migration state.

- MTU of HCA port exceeded.
- Invalid Port.
- Invalid Service Type for this QP
- Maximum number of Work Requests requested exceeds HCA capability.
- Invalid RNR NAK Timer Field value.
- More outstanding entries on WQ than size specified.

11.2.3.3 QUERY QUEUE PAIR

Description:

Returns the attribute list and current values for the specified QP. This QP handle can be any QP handle supplied by the Verbs.

Input Modifiers:

- HCA handle.
- QP handle.

Output Modifiers:

- The QP attributes. The list of attributes returned by the query are:
  - The QP number.
  - Handle of the Completion Queue associated with the Send Queue.
  - Handle of the Completion Queue associated with the Receive Queue.
  - The actual number of outstanding requests supported on the Send Queue.
  - The actual number of outstanding requests supported on the Receive Queue.
  - The actual number of scatter/gather entries supported on Work Requests submitted to the Send Queue.
  - The actual number of scatter/gather entries supported on Work Requests submitted to the Receive Queue.
  - Current QP State. Where the state is one of the following:
    - Reset
    - Initialized
    - Ready to Receive (RTR)
    - Ready to Send (RTS)
    - SQ Error

- SQ Drain (SQD)
  The following modifiers are valid only when the QP is in the SQD state:
  - Send Queue Draining
  - Send Queue Drained
- Error

The following attributes are not defined if the QP is in the Reset state.

- PSNs for Send & Receive QPs. Applicable only for RC & UC QPs.
- RDMA Read enable.
- RDMA Write enable.
- Atomic Operation enable.
- Primary physical port associated with this QP. Not applicable on RD QPs.
- Primary P_Key index. Not applicable for RD & Raw Datagram QPs.
- Q_Key for the Receive Queue. Not applicable to RC, UC & Raw Datagram QPs.
- Reliable Datagram Domain. Applicable only to RD QPs.
- Destination QP number. Applicable only to RC & UC QPs.
- Number of RDMA Reads & Atomic Operations outstanding at any time on the destination QP. Applicable only to RC QPs.
- Number of responder resources for handling incoming RDMA Reads & atomic operations. Applicable only to RC QPs.
- Minimum RNR NAK Timer Field Value. When a message arrives which is targeted at a local receive queue, and that receive queue has no receive work requests outstanding, the CI may respond to the initiator with an RNR NAK packet. This modifier is the minimum value which shall be sent in the Timer Field of such an RNR NAK packet; it does not affect RNR NAKs sent for other reasons. Applicable only to RC and RD QPs.
- Primary Address vector, for RC & UC transports only, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - Path MTU.

- Maximum Static Rate.
- Local ACK Timeout. Applicable only to RC QPs.
- Retry count. Applicable only to RC QPs.
- RNR retry count. Applicable only to RC QPs.
- Source Path Bits.
- For global destination:
  - Traffic class.
  - Flow label.
  - Hop limit.
  - Source GID index.
  - Destination's GID (a.k.a. IPv6) address.

- Alternate path address information, returned only for RC & UC QPs. Valid only when automatic path migration is enabled.
  - Alternate path P_Key index.
  - Alternate path Physical port.

  Path address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.
  - Maximum Static Rate.
  - Local ACK Timeout. Applicable only to RC QPs.
  - Retry count. Applicable only to RC QPs.
  - RNR retry count. Applicable only to RC QPs.
  - Source Path Bits.

    For global destination:
    - Traffic class.
    - Flow label.
    - Hop limit.
    - Source GID index.
    - Destination's GID (a.k.a. IPv6 address).
- Path migration state. Valid only if this HCA supports automatic path migration.

- Verb Results:
  - Operation completed successfully.

- Invalid HCA handle.
- Invalid QP handle.

11.2.3.4 DESTROY QUEUE PAIR

Description:

Destroys the specified QP.

C11-11: Any resources allocated by the Channel Interface in order to process Work Requests on the QP **must** be deallocated as part of the destroy operation.

A QP instance is allowed to have Work Requests outstanding when a request to destroy the QP is made. When a QP is destroyed, any outstanding Work Requests are no longer considered to be in the scope of the Channel Interface. It is the responsibility of the Consumer to clean up resources associated with a Work Request.

C11-12: Outstanding Work Requests on this QP **shall not** be completed after this Verb returns. Incoming operations destined for a QP that has been destroyed are discarded.

Input Modifiers:

- HCA handle.
- QP handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.

11.2.4 GET SPECIAL QP

Description:

Returns the handle for the specified QP type for the specified HCA port. The special QP types include: SMI QP (QP0), GSI QP (QP1), Raw IPv6 and Raw Ethertype.

C11-13: This Verb **must** support QP0 and QP1.

HCA support for both Raw Datagram types is optional.

o11-1: If the HCA supports the Raw Datagram QP types, this Verb **must** also support them.

C11-14: Handles associated with the SMI QP and the GSI QP **must** only be given out once for each QP per HCA port. Subsequent invocations of this Verb, without an intervening Destroy QP, **must** return an error.

The single QP per port restriction does not apply to either Raw Datagram QP types.

o11-2: If Raw Datagram Service is supported, the number of Raw Datagram type QPs supported per port **shall** be returned by the Query HCA Verb.

Any fixed QP attributes for the specified QP type required by the specific implementation are set up before returning from this Verb. For example, the appropriate Transport Service Type may need to be initialized for the QP.

C11-15: SMI/GSI QPs **shall not** share a completion queue with any non-SMI/GSI QP. An attempt to do so **shall** result in an Invalid CQ Handle error.

Input Modifiers:

- HCA Handle.
- HCA port number.
- The QP type requested. The allowed types are:
  - SMI QP (QP0).
  - GSI QP (QP1).
  - Raw IPv6.
  - Raw Ethertype.
- The CQ to be associated with the Send Queue.
- The CQ to be associated with the Receive Queue.
- The maximum number of outstanding Work Requests the Consumer expects to submit to the Send Queue.
- The maximum number of outstanding Work Requests the Consumer expects to submit to the Receive Queue.
- The maximum number of scatter/gather elements the Consumer expects to specify in a Work Request submitted to the Send Queue.
- The maximum number of scatter/gather elements the Consumer expects to specify in a Work Request submitted to the Receive Queue.
- The Signaling Type for the Send Queue on this QP. The valid types are:

    - All Work Requests submitted to the Send Queue always generate a completion entry.
    - Consumer must specify on each Work Request submitted to the Send Queue whether to generate a completion entry for successful completions.
- Protection Domain.

Output Modifiers:

- QP handle.
- The actual number of outstanding Work Requests supported on the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of outstanding Work Requests supported through the Verbs on the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested. (This may require the Consumer to increase the size of the CQ.)
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Send Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- The actual number of scatter/gather elements that can be specified in Work Requests submitted to the Receive Queue. If an error is not returned, this is guaranteed to be greater than or equal to the number requested.
- Verb Results:
    - Operation completed successfully.
    - Insufficient resources to complete request.
    - Invalid HCA handle.
    - Invalid Special QP type.
    - QP already in use (applies only to SMI and GSI QPs).
    - Number of available Raw Datagram QPs exceeded.
    - Invalid Port.
    - Invalid CQ handle.
    - Maximum number of Work Requests requested exceeds HCA capability.
    - Maximum number of scatter/gather elements requested exceeds HCA capability.
    - Invalid Protection Domain.

- Raw Datagrams not supported.

11.2.5 COMPLETION QUEUE

11.2.5.1 CREATE COMPLETION QUEUE

Description:

Creates a CQ on the specified HCA.

The Consumer must specify the minimum number of entries in the CQ.

The actual number of completion entries on the specified CQ is returned on successful creation. The number returned differs only when the number of actual entries is more than the Consumer requested. If the number of entries the HCA supports is less than the Consumer requested, an error is returned.

On success, a handle to the newly created Completion Queue is returned.

Input Modifiers:

- HCA handle.
- The minimum number of entries in the CQ.

Output Modifiers:

- The handle of the newly created CQ.
- The actual number of entries in the CQ.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Number of CQ entries requested exceeds HCA capability.

11.2.5.2 QUERY COMPLETION QUEUE

Description:

Returns the number of entries in the specified CQ.

Input Modifiers:

- HCA handle.
- CQ handle.

Output Modifiers:

- The total number of entries in the CQ.

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.

11.2.5.3 RESIZE COMPLETION QUEUE

Description:

Resizes the CQ.

C11-16: The CI **must** support resizing a CQ with outstanding Work Completions and while Work Requests are outstanding on queues associated with the specified CQ. Completions **must not** be lost as a result of a resize.

The resize operation is allowed to adversely affect the performance while the CQ is being resized. The act of resizing is not allowed to directly generate completion or asynchronous errors.

Input Modifiers:

- HCA handle.
- CQ handle.
- The minimum number of entries in the CQ.

Output Modifiers:

- The actual number of entries in the CQ.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - Number of CQ entries requested exceeds HCA capability.
  - More outstanding entries on CQ than size specified.

11.2.5.4 DESTROY COMPLETION QUEUE

Description:

Destroys the specified CQ. Resources allocated by the Channel Interface to implement the CQ must be deallocated during the destroy operation.

C11-17: The CI **shall** return an error if this Verb is invoked while a Work Queue is still associated with the CQ.

Any completions that have not been retrieved from the CQ prior to being destroyed are discarded.

Input Modifiers:

- HCA handle.
- CQ handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - One or more Work Queues is still associated with the CQ.

11.2.6 EE CONTEXT

11.2.6.1 CREATE EE CONTEXT

Description:

Creates an EE Context for the specified HCA.

On success, a handle to the newly created EE Context is returned.

The values for Remote Node Address Handle, Send Sequence Number, Receive Sequence number are all zero. The EE Context is created in the Reset state.

Input Modifiers:

- HCA handle.
- Reliable Datagram Domain.

Output Modifiers:

- The handle for the newly created EE Context.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Reliable Datagrams not supported.
  - Invalid Reliable Datagram Domain.

11.2.6.2 MODIFY EE CONTEXT ATTRIBUTES

Description:

o11-3: If the CI supports RD Service, upon invocation of this Verb the CI **shall** modify the attributes for the specified EE Context and then **shall** cause the EE Context to transition to the specified EE Context state.

Only a subset of the attributes can be modified once the EE Context has been created.

EE Context attributes can be modified with Work Requests outstanding which use the EE handle, but any such Work Requests might not execute correctly if the modifiers are changed.

WRs can be outstanding on the Receive Queue when the EE is in the Init, RTR, and RTS state and on the Send Queue when the EE is in the RTS state. Any outstanding Work Requests on the QP may not execute as expected if the EE modifiers are changed.

If any invalid attribute or value is specified, an error is returned and all EE Context attributes remain unchanged.

o11-4: If the CI supports RD Service, the properties and requirements of the EE Context state transitions **shall** be supported as shown in Table 80.

Table 80 EE Context State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
Reset to Init	Physical port. P_Key index.	None.	Enable posting to the receive queue.
Init to RTR	Remote Node Address Vector. Destination EEC Number. RQ PSN. Number of responder resources for RDMA Read/atomic ops. (Note this is 1 for this revision.)	Alternate path address information. P_Key index.	Activate receive processing.
RTR to RTS	Local ACK Timeout. Retry count. RNR retry count. SQ PSN. Number of Outstanding RDMA Read/atomic ops at destination.	Alternate path address information. Path migration state.	Activate send processing.
RTS to RTS (no transition)	None.	Alternate path address information. Path migration state.	No transition.
RTS to SQD	None.	None.	Deactivate send processing.

Table 80 EE Context State Transition Properties

Transition	Required Attributes	Optional Attributes	Actions
SQD to RTS	None	Remote Node Address Vector. Alternate path address information. Channel migration state. Timeout/Retry values. Number of Outstanding RDMA Read/atomic ops at destination. Number of local RDMA Read/atomic responder resources. Q_Key. P_Key index. Primary physical port associated with EE if HCA supports this capability.	Activate send processing.
Any State to Error	None.	None allowed.	Queue processing is stopped. Work Requests in process are completed in error, when possible.
Any state to Reset	None.	None allowed.	EE attributes are reset to the same values after the EE was created. Outstanding Work Requests are removed from the queues without notifying the Consumer.

Input Modifiers:

- HCA handle.
- EE Context handle.
- The EE Context attributes to modify and their new values. The EE Context attributes that can be modified after the EE Context has been created are:
  - Primary path Physical Port. Applicable for the SQD to RTS transition, if supported by the HCA.
  - Primary path P_Key Index.
  - PSNs for Sends & Receives.

  - EE Context State.Enable or disable Send Queue Drained, Asynchronous Affiliated Event Notification. This modifier is only applicable when the next EE state chosen is SQD.
- Number of RDMA Reads & Atomic Operations outstanding at any time on the destination EE.
- Number of responder resources for handling incoming RDMA Reads & atomic operations. This value may be rounded up to a supported value, not to exceed the maximum value allowable for EEs for this HCA. Note for this version of the specification, this value is one.
- Destination EE Context number
- Primary Address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
  - Path MTU.
  - Maximum Static Rate.
  - Local ACK Timeout.
  - Retry count.
  - RNR retry count.
  - Source Path Bits.
  - For global destination:
    - Traffic class.
    - Flow label.
    - Hop limit.
    - Source GID index.
    - Destination's GID (a.k.a. IPv6) address.
- Alternate path address information. Valid only when automatic path migration is enabled.
  - Alternate path P_Key index.
  - Alternate path Physical port.

  Alternate path address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.

    - Maximum Static Rate.
    - Local ACK Timeout.
    - Retry count.
    - RNR retry count.
    - Source Path Bits.

    For global destination:

        - Traffic class.
        - Flow label.
        - Hop limit.
        - Source GID index.
        - Destination's GID (a.k.a. IPv6 address).

- Path migration state. Valid only if this HCA supports automatic path migration. Valid states to set are:
    - Migrated.
    - Rearm.

Output Modifiers:

- Verb Results:
    - Operation completed successfully.
    - Insufficient resources to complete request.
    - Invalid HCA handle.
    - Invalid EE Context handle.
    - Cannot change EE Context attribute.
    - Invalid EE Context state.
    - Invalid path migration state.
    - Reliable Datagrams not supported.

11.2.6.3 QUERY EE CONTEXT

Description:

Returns the attribute list and current values for the specified EE Context.

Input Modifiers:

- HCA handle.
- EE Context handle.

Output Modifiers:

- The EE Context attributes. The list of attributes returned by the query are:
  - Current EE Context State. Where the state is one of the following:
    - Reset
    - Initialized
    - Ready to Receive (RTR)
    - Ready to Send (RTS)
    - SQ Error
    - SQ Drain (SQD)
      The following modifiers are valid only when the EE Context is in the SQD state:
      - Send Queue Draining
      - Send Queue Drained
    - Error
  - EE Context Number.

  The following attributes are not defined if the EE is in the Reset state.

  - Primary path Physical Port.
  - Primary path P_Key Index.
  - PSNs for Sends & Receives.
  - Reliable Datagram Domain.
  - Number of RDMA Reads & Atomic Operations outstanding at any time on the destination EE.
  - Number of responder resources for handling incoming RDMA Reads & atomic operations.
  - Destination EE Context number.
  - Primary Address vector, containing:
    - Service level.
    - Send Global Routing Header Flag.
    - Destination LID. If destination is in same subnet, LID = final destination; otherwise LID = router LID.
    - Path MTU.
    - Maximum Static Rate.
    - Local ACK Timeout.
    - Retry count.
    - RNR retry count.

- Source Path Bits.
- For global destination:
  - Traffic class.
  - Flow label.
  - Hop limit.
  - Source GID index.
  - Destination's GID (a.k.a. IPv6) address.
- Alternate path address information. Valid only when automatic path migration is enabled.
  - Alternate path P_Key index.
  - Alternate path Physical port.

  Alternate path address vector, containing:
  - Service level.
  - Send Global Routing Header Flag.
  - Destination LID. If the destination is in the same subnet, LID = final destination; otherwise LID = router LID.
  - Maximum Static Rate.
  - Local ACK Timeout.
  - Retry count.
  - RNR retry count.
  - Source Path Bits.

  For global destination:
    - Traffic class.
    - Flow label.
    - Hop limit.
    - Source GID index.
    - Destination's GID (a.k.a. IPv6 address).
- Path migration state. Applicable only if this HCA supports automatic path migration.

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid EE Context handle.
  - Reliable Datagrams not supported.

11.2.6.4 DESTROY EE CONTEXT

Description:

Destroys the specified EE Context. Any resources allocated by the Channel Interface for use by the EE Context are freed from use.

o11-5: If the CI supports RD Service, after this Verb is invoked, any outstanding or subsequently submitted Work Requests which depend on the EE Context **shall** complete with an Invalid EE Context Number error.

Input Modifiers:

- HCA handle.
- EE Context handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid EE Context handle.
  - Reliable Datagrams not supported.

11.2.7 MEMORY MANAGEMENT

Memory Management Verbs are partitioned into two categories:

1) Registration of memory regions.

   The Verbs used to register memory regions are the following:

   - Register Memory Region.
   - Register Physical Memory Region.
   - Query Memory Region.
   - Deregister Memory Region.
   - Reregister Memory Region.
   - Reregister Physical Memory Region.
   - Register Shared Memory Region.

2) Binding of Memory Windows.

   The Verbs used to allocate and bind Memory Windows are following:

   - Allocate Memory Window.
   - Query Memory Window.
   - Bind Memory Window.
   - Deallocate Memory Window.

11.2.7.1 REGISTER MEMORY REGION

Description:

Prepares a virtually addressed memory region for use by an HCA. A description of the registered memory suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the registered memory suitable for use by inbound RDMA and/or atomic operations is returned.

This Verb depends on OSV supplied functions to perform the pinning of memory pages and creating the virtual to physical translations that represent the memory region.

Input Modifiers:

- HCA Handle.
- Virtual Address - the address of the first byte of the region to be registered. The Maximum size of a Virtual Address is 64 bits.
- Length of region to be registered in bytes.
- Protection Domain to be assigned to the registered region.
- Access Control - The following may be selected in any combination except as noted.
  - Enable Local Write Access.
  - Enable Remote Write Access.

    Remote Write Access requires Local Write Access to be enabled.
  - Enable Remote Read Access.
  - Enable Remote Atomic Operation Access (If Atomic Ops supported).

    Remote Atomic Operation Access requires Local Write Access.
  - Enable Memory Window Binding.

  Note: Local Read Access is always implied.

Output Modifiers:

- Memory Region Handle - used to identify this specific registered region to the Memory Management Verbs.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Verb Results:

- Operation completed successfully.
- Insufficient resources to complete request.
- Invalid HCA handle.
- Invalid Virtual Address.
- Invalid Length
- Invalid Protection Domain.
- Invalid Access Control specifier.

11.2.7.2 REGISTER PHYSICAL MEMORY REGION

Description:

Prepares a physically addressed memory region for use by an HCA. A description of the registered memory suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the registered memory suitable for use by inbound RDMA and/or atomic operations is returned.

In addition to a list of physical buffers, the Consumer supplies a requested “I/O Virtual Address” to be associated with the first byte of the Region. The Consumer also supplies the length of the entire Region plus a byte offset that specifies where the Region begins within the first physical buffer. The Channel Interface returns the I/O Virtual Address that is actually assigned for the Region.

Input Modifiers:

- HCA Handle.
- Physical Buffer List.
  - List of Physical Buffers - Each buffer must begin and end on an HCA-supported page boundary.
  - Total number of Physical Buffers in the list.
- I/O Virtual Address - IOVA requested by the Consumer for the first byte of the region.
- Length of Region to be registered in bytes.
- Offset of Region’s starting IOVA within the first physical buffer.
- Protection Domain to be assigned to the registered region
- Access Control - The following may be selected in any combination except as noted.
  - Enable Local Write Access.
  - Enable Remote Write Access.

    Remote Write Access requires Local Write Access to be enabled.

- Enable Remote Read Access.
- Enable Remote Atomic Operation Access (If Atomic Ops supported).

  Remote Atomic Operation Access requires Local Write Access.
- Enable Memory Window Binding.

Note: Local Read Access is always implied.

Output Modifiers:

- Memory Region Handle - used to identify this specific registered region to the Memory Management Verbs.
- I/O Virtual Address - IOVA actually assigned by the Channel Interface for the first byte of the Region.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Physical Buffer List entry.
  - Invalid Length.
  - Invalid Offset.
  - Invalid Protection Domain.
  - Invalid Access Control specifier.

11.2.7.3 QUERY MEMORY REGION

Description:

Retrieves information about a specific memory region.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.

Output Modifiers:

- L_Key - as issued when region was registered.
- R_Key - as issued when region was registered.

- Actual Local Protection Bounds enforced by the Channel Interface.
- Actual Remote Protection Bounds enforced by the Channel Interface.

  The Remote Protection Bounds are returned only when Local and Remote Access to the region was requested.
- Protection Domain assigned to the registered region.
- Access Control settings for the registered region.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle
  - Invalid Memory Region handle.

11.2.7.4 DEREGISTER MEMORY REGION

Description:

Removes a memory region from the HCA translation table. The region is unpinned if pinned in the associated registration Verb. This Verb is responsible only for deallocating resources allocated as part of the associated registration operation. All other resources are the responsibility of the Consumer.

It is an error for a Consumer to attempt to deregister a Memory Region while it still has any Memory Windows bound to it. Channel Interface implementations have options on how to deal with the error, described in 10.6.6.2.4 Deregistering Regions with Bound Windows on page 440.

Work Requests or Remote Operation requests that are in process and actively referencing memory locations in a Memory Region that is deregistered must fail with a protection violation. Work Requests or Remote Operation requests that attempt to access memory locations in a Memory Region that has been deregistered must fail with a protection violation.

This Verb depends on the availability of OSV supplied functions to perform the unpinning of memory pages.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.

Output Modifiers:

No output modifiers.

- Verb Results:

- Operation completed successfully.
- Invalid HCA handle.
- Invalid Memory Region handle.
- Operation denied; Region still has bound Window(s)

11.2.7.5 REREGISTER MEMORY REGION

Description:

Modifies the attributes of an existing Memory Region. Any existing Region owned by the Consumer can be modified, regardless of which Verb created it initially[1], or which Verb (if any) reregistered it most recently[2]. A description of the Memory Region suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the Memory Region suitable for use by inbound RDMA and/or atomic operations is returned.

This Verb conceptually performs the functions Deregister Memory Region followed by Register Memory Region. Where possible, resources below the Verb layer are expected to be reused instead of deallocated and reallocated. This Verb may be used to change the access rights and/or protection domain of a region, as well as changing the memory locations that are registered.

The L_Key and R_Key output modifiers from this Verb must be used in place of any previously issued for this region.

This Verb depends on the availability of OSV supplied functions to perform the pinning and unpinning of memory pages and creating the virtual to physical translations that represent the memory region.

It is an error for a Consumer to attempt to reregister a Memory Region while the Region still has any Memory Windows bound to it. Channel Interface implementations have options on how to deal with the error, as described in 10.6.6.2.4 Deregistering Regions with Bound Windows on page 440.

C11-18: If the CI returns the "Operation denied" (due to a bound Window) error, the CI **shall** make no change to the current registration.

C11-19: If the CI returns either the "Invalid HCA handle" or "Invalid Memory Region handle" error, the CI **shall** make no change to the current registration (assuming that it even exists).

C11-20: If the CI returns any other error, the CI **shall** invalidate both "old" and "new" registrations, and release any associated resources.

1. For instance, a Region created with Register *Physical* Memory Region can later be modified by Reregister Memory Region.

2. For instance, a Region modified by Reregister *Physical* Memory Region can later be modified by Reregister Memory Region.

C11-21: For the error case where a remote agent is accessing a Memory Region while it is in the process of being reregistered, the CI **must** present the same semantics as a deregistration operation followed by a separate registration operation.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.
- Change Request type - The following may be selected in any combination, the input modifiers required to support the request are listed below each request.
  - Change Translation.

    Input Modifiers required.
    - Virtual Address.
    - Length.
  - Change Protection Domain.

    Input Modifiers required.
    - Protection Domain.
  - Change Access Control.

    Input Modifiers required.
    - Access Control Selections.

Output Modifiers:

- Memory Region Handle - must be used for future references to this Memory Region. Might or might not be the same as the previous Region Handle.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Memory Region handle.
  - Invalid Virtual Address.
  - Invalid Length.
  - Invalid Protection Domain.

- Invalid Access Control specifier.
- Operation denied; Region still has bound Window(s)

Usage Example:

a) To modify only the Access Control of an already registered region, the Memory Region Handle, a Change Access Control Request and the new Access Control Selections input modifiers would be supplied to the Verb.

b) To change the address translations of a region the Memory Region Handle, a Change Translation Request and the new Virtual Address and length input modifiers would be supplied to the Verb. The pages previously pinned would be unpinned, the new memory region would be pinned and registered (and if requested bound) using the region's access controls and protection domain. Previous translations would be removed or replaced as needed.

11.2.7.6 REREGISTER PHYSICAL MEMORY REGION

Description:

Modifies the attributes of an existing Memory Region. Any existing Region owned by the Consumer can be modified, regardless of which Verb created it initially[1], or which Verb (if any) reregistered it most recently[2]. A description of the Memory Region suitable for use in Work Requests to describe locally accessible memory locations is returned. When specifically requested, a description of the Memory Region suitable for use by inbound RDMA and/or atomic operations is returned.

This Verb conceptually performs the functions Deregister Memory Region followed by Register Physical Memory Region. Where possible, resources below the Verb layer are expected to be reused instead of deallocated and reallocated. This Verb may be used to change the access rights and/or protection domain of a region, as well as changing the memory locations that are registered.

The L_Key and R_Key output modifiers from this Verb must be used in place of any previously issued for this region.

It is an error for a Consumer to attempt to reregister a Memory Region while the Region still has any Memory Windows bound to it. Channel Interface implementations have options on how to deal with the error, as described in 10.6.6.2.4 Deregistering Regions with Bound Windows on page 440.

1. For instance, a Region created with Register Memory Region can later be modified by Reregister *Physical* Memory Region.

2. For instance, a Region modified by Reregister Memory Region can later be modified by Reregister *Physical* Memory Region.

C11-22: For the Reregister Physical Memory Region Verb, the CI **shall** conform to all of the compliance statements contained in 11.2.7.5 Reregister Memory Region .

Input Modifiers:

- HCA Handle.
- Memory Region Handle - as issued when region was registered.
- Change Request type - The following may be selected in any combination, the input modifiers required to support the request are listed below each request.
  - Change Translation.

    Input Modifiers required.
    - Physical Buffer List.
      - List of Physical Buffers - Each buffer must begin and end on an HCA-supported page boundary.
      - Total number of Physical Buffers in the list.
    - I/O Virtual Address - IOVA requested by the Consumer for the first byte of the region.
    - Length of Region to be registered in bytes.
    - Offset of Region's starting IOVA within the first physical buffer.
  - Change Protection Domain.

    Input Modifiers required.
    - Protection Domain.
  - Change Access Control.

    Input Modifiers required.
    - Access Control Selections.

Output Modifiers:

- Memory Region Handle - must be used for future references to this Memory Region. Might or might not be the same as the previous Region Handle.
- I/O Virtual Address - IOVA actually assigned by the Channel Interface for the first byte of the Region.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned only when Remote Access was requested.
- Verb Results:

- Operation completed successfully.
- Insufficient resources to complete request.
- Invalid HCA handle.
- Invalid Memory Region handle.
- Invalid Virtual Address.
- Invalid Length.
- Invalid Offset.
- Invalid Protection Domain.
- Invalid Access Control specifier.
- Operation denied; Region still has bound Window(s)

11.2.7.7 REGISTER SHARED MEMORY REGION

Description:

Given an existing Memory Region, a new independent Memory Region associated with the same physical memory locations is created, with the intention that the new Memory Region share HCA mapping resources to the extent possible. Through repeated calls to the Verb, an arbitrary number of Memory Regions can potentially share the same HCA mapping resources, all associated with the same physical memory locations. The memory region created by this verb behaves identically to memory regions created by the other memory registration verbs.

The Virtual Address, Protection Domain, and Access Rights specified for the new Memory Region need not be the same as those of the existing Memory Region. The lengths are by definition the same.

The Consumer supplies a requested Virtual Address to be associated with the first page in the new Memory Region, and the Channel Interface returns the Virtual Address that is actually assigned.

Input Modifiers:

- HCA Handle.
- Memory Region Handle - of an already registered region.
- Virtual Address - requested by the Consumer for the first page of the buffer.
- Protection Domain.
- Access Control Selections.

Output Modifiers:

- Memory Region Handle - of the new Memory Region.

- Virtual Address - actually assigned by the Channel Interface for the first page.
- L_Key - used for local access.
- R_Key - used for remote access.

  The R_Key is returned when Remote Access Rights are requested.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Memory Region handle.
  - Invalid Protection Domain.
  - Invalid Access Control specifier.

11.2.7.8 ALLOCATE MEMORY WINDOW

Description:

This Verb allocates a memory window which is associated with a protection domain. It is not inherently associated with any memory region when allocated.

Input Modifiers:

- HCA Handle.
- Protection Domain to be assigned to the Memory Window.

Output Modifiers:

- Window Handle - used to identify this specific Memory Window to other Memory Management Verbs.
- R_Key - an unbound R_Key for use in specifying the Window with the Bind Memory Window Verb.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle.
  - Invalid Protection Domain.

11.2.7.9 QUERY MEMORY WINDOW

Description:

This Verb returns the attributes associated with the specified memory window.

Input Modifiers:

- HCA Handle.
- Window Handle - as issued by an Allocate Memory Window.

Output Modifiers:

- R_Key - the current R_Key associated with the Memory Window.
- Protection Domain associated with the Memory Window.
- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid Memory Window handle.

11.2.7.10 BIND MEMORY WINDOW

Description:

Posts a Work Request to a specified Send Queue, which binds a Memory Window to a specified VA range and remote access attributes based on an existing Memory Region. The QP Service Type must be either Reliable Connection, Unreliable Connection, or Reliable Datagram.

The specified VA range must either be the entire Memory Region or a subset of it. Remote Write Access or Remote Atomic Access must not be specified unless the Memory Region has Local Write Access. The QP, Memory Window, and Memory Region must belong to the same HCA and Protection Domain.

A previously bound Memory Window can be bound to a new VA range in the same or a different Memory Region, causing the previous binding to be invalidated. Binding a previously bound Memory Window to a zero-length VA range will invalidate the previous binding and return an R_Key that is in the unbound state.

The Bind operation has a unique ordering rule: any Work Request posted to a Send Queue subsequent to a Bind must not begin execution until the Bind operation completes.

Under normal operation, it is improper for a Consumer to change the binding of a Memory Window while it is being accessed by a remote agent. However, this can occur if remote agents misbehave, or it can occur under error recovery circumstances. Any Remote Operation requests that are in process and actively using a Memory Window when its binding is changed must fail with a protection violation. Once the Bind operation has been reported to the Consumer as having completed, the Channel Interface must guarantee that no additional accesses can be performed under the immediate previous binding.

Input Modifiers:

- HCA Handle.
- QP Handle.
- The Work Request containing the information required to perform the request. The Work Request is defined as follows:
  - A user defined 64-bit Work Request ID.
  - Memory Window Handle.
  - R_Key - The R_Key currently associated with the Memory Window.
  - Memory Region Handle.
  - L_Key - The L_Key for the Memory Region that the Memory Window will be associated with.
  - Virtual Address - the address of the first byte of the bound range. The Maximum size of a Virtual Address is 64 bits.
  - Length of range to be bound in bytes.
  - Access Control - The following may be selected in any combination except as noted.
    - Enable Remote Write Access.
    - Requires the Memory Region to have Local Write Access.
    - Enable Remote Read Access
    - Enable Remote Atomic Operation Access (If Atomic Ops supported)
  - Requires the Memory Region to have Local Write Access.
  - Completion notification type. Must be specified and is only valid if the Send Queue was set up for selectable signaling.

Output Modifiers:

- R_Key - The R_Key associated with the new binding, whose value is different from that of the supplied R_Key.
- Verb Results:
  - Operation completed successfully.
  - Insufficient resources to complete request.
  - Invalid HCA handle
  - Invalid QP handle.

    - Invalid Service Type for this QP.
    - Invalid Memory Window handle.
    - Invalid R_Key.
    - Invalid Memory Region handle.
    - Invalid L_Key.
    - Invalid Virtual Address
    - Invalid Length.
    - Invalid Access Control specifier.
    - Invalid completion notification type.
    - Too many Work Requests posted.
- Work Request Completion Status
    - Operation completed successfully.
    - Protection Error.

11.2.7.11 DEALLOCATE MEMORY WINDOW

Description:

Under normal operation, it is improper for a Consumer to deallocate a Memory Window while it is being accessed by a remote agent. However, this can occur if remote agents misbehave, or it can occur under error recovery circumstances. Any Remote Operation requests that are in process and actively using a Memory Window when it is deallocated must fail with a protection violation. Once the deallocation Verb completes, the Channel Interface must guarantee that no additional accesses can be performed through that Memory Window.

Input Modifiers:

- HCA Handle.
- Window Handle - as issued by an Allocate Memory Window.

Output Modifiers:

No output modifiers

- Verb Results:
    - Operation completed successfully.
    - Invalid HCA handle
    - Invalid Memory Window handle.

11.3 MULTICAST

11.3.1 ATTACH QP TO MULTICAST GROUP

Description:

Attaches the QP to the specified multicast group. The only function of this Verb is to assign the Receive Work Queue of this QP to the specified multicast group; after the attachment completes, this QP will be provided with a copy of every multicast message addressed to the specified group and received on the HCA port with which the QP is associated. Creation of the multicast group, and reconfiguration of the fabric such that packets addressed to that group are routed to a local HCA port, is described in 7.10 IBA and Raw Packet Multicast on page 186.

The Service Type of the specified QP must be Unreliable Datagram. It is an error to specify a QP with any other Service Type.

One or more QPs are allowed to be attached to a multicast group on the HCA. If the maximum number of multicast group attachments has already been reached for the HCA when a QP attempts to attach to the multicast group, an error is returned.

The input modifier which determines the multicast group to attach to can be either a DLID, an IPv6 Address or both.

The IBA unreliable multicast feature is optional. This Verb is required only if IBA unreliable multicast is supported by the HCA.

Input Modifiers:

- HCA Handle.
- Multicast group DLID.
- Multicast group IPv6 Address.
- QP Handle.

Output Modifiers:

- Verb Results:
    - Operation completed successfully.
    - Insufficient resources to complete request.
    - Invalid HCA handle.
    - Invalid multicast DLID.
    - Invalid Multicast group IPv6 Address.
    - Invalid QP handle.
    - Invalid Service Type for this QP.
    - Number of QPs attached to multicast groups exceeded.

11.3.2 DETACH QP FROM MULTICAST GROUP

Description:

Detaches the specified QP from a multicast group. The only function of this Verb is to detach the Receive Work Queue of this QP from the specified multicast group.

All of the input modifiers must be correct for the QP to be detached. If the QP is attached to a different multicast group or port, an error will be returned.

This Verb is required only if IBA unreliable multicast is supported by the HCA. The IBA unreliable multicast feature is optional.

Input Modifiers:

- HCA Handle.
- Multicast group DLID.
- Multicast group IPv6 Address.
- QP Handle.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid HCA port number.
  - Invalid multicast DLID.
  - Invalid Multicast group IPv6 Address
  - Invalid QP handle.

11.4 WORK REQUEST PROCESSING

11.4.1 QUEUE PAIR OPERATIONS

11.4.1.1 POST SEND REQUEST

Description:

Builds a WQE for the Send Queue in the specified QP from the information contained in the Work Request submitted by the Consumer. This WQE is added to the end of the Send Queue and the HCA is notified that a new WQE is ready to be processed.

If the Send Queue is enabled for selectable completion notification, the Consumer must specify whether a successful completion of this Work Request results in a completion entry on the CQ.

Control returns to the Consumer immediately after the WQE has been submitted to the Send Queue and the HCA has been notified that a new WQE is ready to process. When control returns, the Work Request is in the scope of the Consumer and will no longer be modified or accessed below the Channel Interface.

C11-23: The CI **shall** return control to the Consumer immediately after the Work Request has been submitted to the Send Queue.

C11-24: Once control has been returned to the Consumer the CI **shall not** modify or access the Work Request.

Sends, RDMA and atomic operations can all take place on the same QP. Operation Type Matrix shows which operations are allowed for each Service Type of the QP.

C11-25: The CI **shall** support the operations based on QP Service type according to Operation Type Matrix.

Table 81 Operation Type Matrix

	Send	RDMA Read	RDMA Write	Atomic Ops
Reliable Connected	Yes	Yes	Yes	Yes
Reliable Datagram	Yes	Yes	Yes	Yes
Unreliable Connected	Yes	Not allowed	Yes	Not allowed
Unreliable Datagram	Yes	Not allowed	Not allowed	Not allowed
Raw Datagram	Yes	Not allowed	Not allowed	Not allowed

The ordering and fencing considerations for Atomic Operations are the same as for RDMA Read.

Not all of the Input Modifiers are valid for all operations. Work Request Modifier Matrix shows which of the Input Modifiers are valid for each operation. If Input Modifiers are specified that are not valid for a particular operation, they are ignored.

C11-26: The CI **shall** ignore all input modifiers in a Work Request that are not valid for the specified operation as shown in Work Request Modifier Matrix.

Table 82 Work Request Modifier Matrix[a]

	Send	RDMA Read	RDMA Write	Atomic Ops
Work Request ID	Required	Required	Required	Required
Completion notification indicator	Required if Send Queue selectable signaled	Required if Send Queue selectable signaled	Required if Send Queue selectable signaled	Required if Send Queue selectable signaled
Scatter/Gather list	Required[b]	Required[b]	Required[b]	N/A
# of Data Segments	Required[b]	Required[b]	Required[b]	N/A
Immediate Data	Optional except N/A for Raw Datagram QPs	N/A	Optional	N/A
Fence Indicator	Optional for Reliable QPs	Optional for Reliable QPs	Optional for Reliable QPs	Optional for Reliable QPs
Remote Node Address	Address Handle Required for UD QPs, DLID, Source Path Bits & SL Required for Raw	N/A	N/A	N/A
Remote Node QP # and Q_Key	Required for IB Datagram QPs	Required for Reliable Datagram QPs	Required for Reliable Datagram QPs	Required for Reliable Datagram QPs
EE Context	Required for Reliable Datagram QPs	Required for Reliable Datagram QP	Required for Reliable Datagram QP	Required for Reliable Datagram QP
Remote address and R_Key	N/A	Required	Required	Required
Atomic operands	N/A	N/A	N/A	Required
Solicited Event	Optional	N/A	Optional with Immediate Data	N/A
Ethertype	Required for Raw Ethertype QPs	N/A	N/A	N/A

Table 82 Work Request Modifier Matrix[a]

	Send	RDMA Read	RDMA Write	Atomic Ops
Maximum Static Rate	Required for Raw and N/A for others	N/A	N/A	N/A

a. Note: If the Service Type is not mentioned in a field in the above table, the modifier is not applicable for that Service Type.
b. Scatter/Gather list is allowed to have zero elements.

Input Modifiers:

This is the full list of modifiers for all of the operations available on the Send Queue. Not all modifiers can be used for all queue or operation types. See Operation Type Matrix and Work Request Modifier Matrix for details on which modifiers may be used for the specified queue and operation types.

- HCA handle.
- QP handle.
- The Work Request containing the information required to perform the request. The modifiers that must be specified are dependent on the operation type specified. The Work Request is defined as follows:
  - A user defined 64-bit Work Request ID.
  - Operation type. Valid operation types for Work Requests submitted to the Send Queue are:
    - Send
    - RDMA Read
    - RDMA Write
    - Compare & Swap (assuming the HCA supports atomic operations)
    - Fetch & Add (assuming the HCA supports atomic operations)
  - Completion notification type. Must be specified and is only valid if the Send Queue was set up for selectable signaling.
  - Scatter/Gather list. The scatter/gather list can contain zero or more Data Segments. The list is specified only for Send and RDMA operations. Note that for Raw IPv6 QPs, the first 40 bytes of the buffer(s) referred to by the Scatter/Gather list must contain the IPv6 header of the outgoing message.
  - Number of Data Segments in the scatter/gather list. This modifier is used only when the scatter/gather list must be specified.

- Immediate Data Indicator. This is set if Immediate Data is to be included in the outgoing request. Valid only for Send or Write RDMA operations.
- 4-byte Immediate Data. Valid only for Send or Write RDMA operations.
- Fence indicator. If the fence indicator is set, then all prior RDMA Read and Atomic Work Requests on the queue must be completed before starting to process this Work Request. The Fence indicator only has an effect with the Reliable Connection and Reliable Datagram transport services.
- Remote node address, required only for operations on Raw or IB Unreliable Datagram Service Types.
- QP number of the destination QP. Required only for operations on IB Datagram Service Types.
- The Q_Key for the destination QP. Required only for operations on IB Datagram Service Types. See 10.2.4 Q_Keys on page 403 for more detail on how the CI determines which Q_Key to insert in the packet.
- Ethertype associated with the Work Request. Required only for Raw Ethertype QPs.
- Maximum Static Rate. Required only for Raw IPv6 and Raw Ethertype QPs.
- EE Context. Required only for Reliable Datagram QPs. Note that this is the EE Context number and not the EE Context Handle.
- Solicited Event Indicator. Valid only for Sends or RDMA Writes with immediate data.
- Remote address specified by an address and R_Key. Required and used only for RDMA and atomic operations. For Atomic operations, the address must point to a location that is 64-bit aligned.
- Atomic operation operands. If an atomic operation is specified, the following additional operands must be supplied:
  - 1st 64-bit operand. Must be aligned on a 64-bit boundary. It is the value to compare against for Compare & Swap. The value to add for Fetch & Add.
  - 2nd 64-bit operand. Must be aligned on a 64-bit boundary. This value replaces the previous contents of the remote address if the first argument equals the content of the location in the Compare & Swap. Ignored for Fetch & Add.

- A local Data Segment where a copy of the original contents of the remote memory operation will be deposited after the atomic operation completed at the remote endnode.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.
  - Too many Work Requests posted.
  - Invalid operation type.
  - Invalid QP state.

    Note: This error is returned only when the QP is in the Reset, Init, or RTR states. It is not returned when the QP is in the Error state due to race conditions that could result in indeterminate behavior. Work Requests posted to the Send Queue while the QP is in the Error state are completed with a flush error.
  - Invalid completion notification type.
  - Invalid Scatter/Gather list format.
  - Invalid Scatter/Gather list length.
  - Atomic operations not supported.
  - Invalid address handle.

11.4.1.2 POST RECEIVE REQUEST

Description:

Builds a WQE for the Receive Queue in the specified QP from the information contained in the Work Request submitted by the Consumer. This WQE is added to the end of the Receive Queue and the HCA is notified that a new WQE is ready to be processed.

Control returns to the Consumer immediately after the WQE has been submitted to the Receive Queue and the HCA has been notified that a new WQE is ready to process. When control returns, the Work Request is in the scope of the Consumer and will no longer be modified or accessed below the Channel Interface.

C11-27: The CI **shall** return control to the Consumer immediately after the Work Request has been submitted to the Receive Queue.

Input Modifiers:

- HCA handle.

- QP handle.
- The Work Request containing the information required to perform the request. The modifiers that must be specified are dependent on the operation type specified. The Work Request is defined as follows:
  - A user defined 64-bit Work Request ID.
  - Operation type. The only valid operation for the Receive Queue is the Receive operation.
  - Scatter/Gather list. This list can contain zero or more Data Segments.

    Note that for UD QPs, the first 40 bytes of the buffer(s) referred to by the Scatter/Gather list will contain the GRH of the incoming message. If no GRH is present, the contents of first 40 bytes of the buffer(s) will be undefined. The presence of the GRH will be indicated by a bit in the Work Completion.
  - Number of Data Segments in the scatter/gather list.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid QP handle.
  - Too many Work Requests posted.
  - Invalid operation type.
  - Invalid QP state.
  - Invalid Scatter/Gather list format.
  - Invalid Scatter/Gather list length.

11.4.2 Completion Queue Operations

11.4.2.1 Poll for Completion

Description:

Polls the specified CQ for a Work Completion. A Work Completion indicates that a Work Request for a Work Queue associated with the CQ is done.

If an entry is present, the Work Completion at the head of the CQ is returned to the Consumer.

The following table defines, classifies and associates wire level protocol NAK codes with completion errors that are possible on Work Requests posted to the Send Queue. Completion errors are returned through the completion queue as work completions.

C11-28: The CI **shall** return completion errors for a Work Request in the associated Work Completion for errors described in Completion Error Types for Send Queues.

Table 83 Completion Error Types for Send Queues

Error Type	Completion Type	Transport Errors returned by responder (RC)	Transport Errors sent by responder (RD)
Local Length	Interface	N/A	N/A
Local Operation	Interface	N/A	N/A
Local Operation	Processing	N/A	Optional NAK - Invalid Request
Local Protection	Interface	N/A	N/A
Local Protection	Processing	N/A	Optional NAK - Invalid Request
Work Request Flushed	Processing	N/A	N/A
Memory Window Bind	Interface	N/A	N/A
Remote Access	Processing	NAK - Remote Access Violation	NAK - Remote Access Violation
Remote Operation	Processing	NAK - Remote Operational Error	NAK - Remote Operational Error
Remote Invalid Request	Processing	NAK - Invalid Request	NAK - Invalid Request
Remote Invalid RD Request	Processing	N/A	NAK - Invalid RD Request
RNR NAK Counter Exceeded	Processing	NAK - RNR	NAK - RNR
Transport Timeout Retry Count Exceeded	Processing	N/A	N/A

A Remote Q_Key violation and a Remote RDD Mismatch will both result in an Invalid RD Request completion error type for the requester's WQE. Since the same NAK code is returned in both cases, it is not possible for the requester to distinguish between them.

The following table defines, classifies and associates wire level protocol NAK codes with completion errors that are possible on Work Requests posted to the Receive Queue. Completion errors are returned through the completion queue as work completions.

C11-29: The CI **shall** generate the completion errors based on the NAK codes as shown in Completion Error Types for Receive Queues.

Table 84 Completion Error Types for Receive Queues

Error Type	Completion Type	Transport Errors sent to Requester (RC and RD)
Local Length	Processing	NAK - Remote Operational Error
Local Protection	Processing	NAK - Remote Operational Error
Local Operation	Processing	NAK - Remote Operational Error

Input Modifiers:

- HCA handle.
- CQ handle.

Output Modifiers:

- The Work Completion containing information relating to the completed Work Request if an entry is present on the CQ. If the status of the operation that generates the Work Completion is anything other than success, the contents of the Work Completion are undefined except as noted below. The contents of a Work Completion are:
  - The 64-bit Work Request ID set by the Consumer in the associated Work Request. This is always valid, regardless of the status of the operation.
  - The operation type specified in the completed Work Request.
  - The valid operation types are:
    - Send (for WRs posted to the Send Queue)
    - RDMA Write (for WRs posted to the Send Queue)
    - RDMA Read (for WRs posted to the Send Queue)
    - Compare and Swap (for WRs posted to the Send Queue)
    - Fetch and Add (for WRs posted to the Send Queue)
    - Memory Window Bind (for WRs posted to the Send Queue)
    - Send Data Received (for WRs posted to the Receive Queue)
    - RDMA with Immediate Data Received (for WRs posted to the Receive Queue)
  - The number of bytes transferred.

The number of bytes transferred is returned in Work Completions for Receive Work Requests for incoming Sends and RDMA Writes with Immediate Data. This does not include the length of any immediate data.

The number of bytes transferred is returned in Work Completions for Send Work Requests for RDMA Read and Atomic Operations.

In the case of UD QPs, the number of bytes transferred is the payload of the message plus the 40 bytes reserved for the GRH. The 40 bytes is always included, whether or not the GRH is present.

- Immediate data indicator. This is set if immediate data is present.
- 4-byte immediate data.
- Remote node address and QP. Returned only for Datagram services. The address information returned for incoming Datagrams is shown in Datagram addressing information.
- GRH Present indicator, for UD QPs only. If this indicator is set, the first 40 bytes of the buffer(s) referred to by the Scatter/Gather list will contain the GRH of the incoming message. If it is not set, the contents of first 40 bytes of the buffer(s) will be undefined. Contents of the payload of the message will begin after the first 40 bytes.

Table 85 Datagram addressing information

Reliable Datagrams	Unreliable Datagrams	Raw IPv6	Raw Ethertype
16-bit SLID	16-bit SLID	16-bit SLID	16-bit SLID
4-bit SL	4-bit SL	4-bit SL	4-bit SL
24-bit Source QP	24-bit Source QP		16-bit Ethertype
24-bit local EE Number	DLID Path Bits	DLID Path Bits	DLID Path Bits

- P_Key index, for GSI only.
- Status of the operation. This is always valid.
  - These status codes are covered in Completion Return Status, with NAK codes reported according to Completion Error Types for Send Queues and Completion Error Types for Receive Queues.
- Freed Resource Count (see 10.8.5.1 Freed Resource Count on page 452). This is always valid, regardless of the status of the operation.

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - CQ empty.

11.4.2.2 REQUEST COMPLETION NOTIFICATION

Description:

Requests the CQ event handler be called when the next completion entry of the specified type is added to the specified CQ. The handler is called at most once per Request Completion Notification call for a particular CQ. Any CQ entries that existed before the notify is enabled will not result in a call to the handler.

Completion Events are one of two types: solicited or unsolicited. A Solicited Completion Event occurs when an incoming Send or RDMA Write with Immediate Data message, with the Solicited Event header bit set causes a successful Receive Work Completion to be added to a CQ; or, when an unsuccessful Work Completion is added to a CQ. An Unsolicited Completion Event occurs when any other successful Receive Work Completion, or any successful Send Work Completion, is added to a CQ.

C11-30: The CI **shall** support both solicited and unsolicited Completion Event Types.

When the Consumer requests completion notification, it must specify whether the notification callback is invoked for either:

- the next Solicited Completion Event only, or
- the next Solicited or Unsolicited Completion Event.

If a Request Completion Notification is pending, subsequent calls to Request Completion Notification for the same CQ prior to the completion event affect only when the notification occurs. A Request Completion Notification for the next completion event takes precedence over a Request Completion Notification for a solicited event completion for the same CQ.

If multiple calls to Request Completion Notification have been made for the same CQ and at least one of the requests set the type to the next completion, the CQ event handler will be called when the next completion is added to that CQ. The CQ event handler will be called only once, even though multiple CQ notification requests were made prior to the completion event for the specified CQ.

Once the CQ event handler is called, another completion notification request must be registered before the CQ event handler will be called again.

C11-31: When a completion notification request is outstanding on a CQ for a *solicited* completion type and another request for that CQ is made that specifies a notification for the *next* completion, the CI **shall** change the outstanding completion notification type to the *next* completion.

C11-32: When a completion notification request is outstanding on a CQ for the *next* completion and another notification request for that CQ is made, the CI **shall not** change the outstanding completion notification type.

A CQ event handler must be specified prior to calling this routine. If the CQ event handler has not been registered when the event is generated, the handler call will not be made.

When the CQ event handler is called, it only indicates a new entry was added to the specified CQ. The HCA and CQ handles are passed to the CQ event handler so the CQ event handler can determine which CQ caused it to be called.

Once the handler routine has been invoked, the Consumer must call Request Completion Notification again to be notified when a new entry is added to that CQ.

It is the responsibility of the Consumer to call the Poll for Completion Verb to retrieve a Work Completion.

Input Modifiers:

- HCA handle.
- CQ handle.
- Type of completion notification requested. The type is either the next completion or when a solicited completion occurs.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.
  - Invalid CQ handle.
  - Invalid completion notification type.

11.5 EVENT HANDLING

11.5.1 SET COMPLETION EVENT HANDLER

Description:

Registers a CQ event handler. Only one CQ event handler can be registered per HCA. Additional calls to this Verb will overwrite the handler routine to be called. Additional calls will not generate an additional handler routine.

This call does not automatically request a notification on a completion event. The Request Completion Notification Verb must be called in order to request notification.

The parameters passed to the CQ event handler are:

- HCA handle.
- CQ handle.

Input Modifiers:

- HCA handle.
- Handler address to call.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.

11.5.2 SET ASYNCHRONOUS EVENT HANDLER

Description:

Registers the asynchronous event handler. Only one asynchronous event handler can be registered per HCA. Additional calls to this Verb will overwrite the handler routine to be called. Additional calls will not generate an additional handler routine.

C11-33: The CI **shall** use the asynchronous event handler specified in this Verb even in the case where an existing asynchronous event handler has already been registered.

After the asynchronous event handler is registered, all subsequent asynchronous events will result in a call to the handler. Until an asynchronous event handler is registered, asynchronous events will be lost.

The parameters passed to the asynchronous event handler are:

- HCA handle.
- Event record. This contains information which indicates the resource type and identifier as well as which event occurred. See Asynchronous Events for more information.

Input Modifiers:

- HCA handle.
- Handler address.

Output Modifiers:

- Verb Results:
  - Operation completed successfully.
  - Invalid HCA handle.

11.6 RESULT TYPES

11.6.1 IMMEDIATE RETURN RESULTS

This section contains a list of the possible Verb return results. All results except "Operation completed successfully" are due to interface errors in the Immediate Error category. Not all Verbs return all results.

Successful return result:

- Operation completed successfully.

Resource errors:

- Insufficient resources to complete request.
- Number of CQ entries requested exceeds HCA capability.
- Maximum number of Work Requests requested exceeds HCA capability.
- Maximum number of scatter/gather elements requested exceeds HCA capability.
- Too many Work Requests posted.
- Number of available Raw Datagram QPs exceeded.
- Number of QPs attached to multicast groups exceeded.
- HCA already in use.

HCA attribute errors:

- Invalid HCA name.
- Invalid HCA handle.
- MTU of HCA port exceeded.
- Invalid Port.
- Invalid Counter specified.
- Invalid Counter value.

Address errors:

- Invalid Address handle.

QP errors:

- Invalid QP handle.
- Cannot change QP attribute.

- Invalid QP state.
- Invalid Service Type for this QP.
- QP is already in use.
- Atomic operations not supported.
- Raw Datagrams not supported.
- Reliable Datagrams not supported.
- Invalid operation type.
- Invalid Scatter/Gather list format.
- Invalid Scatter/Gather list length.
- Invalid path migration state.
- Invalid Special QP type.
- Invalid Address Handle
- More outstanding entries on WQ than size specified.

CQ errors:

- Invalid CQ handle.
- More outstanding entries on the CQ than size specified.
- One or more Work Queues still associated with the CQ.
- CQ empty.
- Invalid completion notification type.

EE Context errors:

- Invalid EE Context handle.
- Invalid EE Context state.
- Cannot change EE Context attribute.

QP or EE Context errors:

- Invalid path migration state.
- Reliable Datagram Domain is in use.
- Invalid Reliable Datagram Domain.
- Invalid RNR NAK Timer Field value.

Memory operation errors:

- Invalid Protection Domain.
- Protection Domain is in use.
- Invalid Virtual Address.
- Invalid Length.
- Invalid Physical Buffer List entry.
- Invalid Offset.

- Invalid L_Key.
- Invalid R_Key.
- Invalid Physical Buffer List entry.
- Invalid Memory Region handle.
- Invalid Memory Window handle.
- Invalid Access Control specifier.
- Operation denied; Region still has bound Window(s)

Multicast errors:

- Invalid multicast DLID.
- Invalid Multicast group IPv6 Address.

Partition table errors:

- P_Key index out of range.
- P_Key index specifies invalid entry in the P_Key table.

11.6.2 COMPLETION RETURN STATUS

Describes the possible Work Completion status error return results. These are errors that occur during the processing of a Work Request and can be reported in the Work Completion status.

- Success - Operation completed successfully.
- Local Length Error - Generated for a Work Request posted to the local Send Queue when the sum of the Data Segment lengths exceeds the message length for the channel. Generated for a Work Request posted to the local Receive Queue when the sum of the Data Segment lengths is too small to receive a valid incoming message.
- Local QP Operation Error - An internal QP consistency error was detected while processing this Work Request.
- Local EE Context Operation Error - An internal EE Context consistency error was detected while processing this Work Request.
- Local Protection Error - The locally posted Work Request's Data Segment does not reference a Memory Region that is valid for the requested operation.
- Work Request Flushed Error - A Work Request was in process or outstanding when the QP transitioned into the Error State.
- Memory Window Bind Error - The Verbs Consumer had insufficient access rights.

The following errors are reported only for Reliable QPs.

- Remote Invalid Request Error - The responder detected an invalid message on the channel. Possible causes include the operation is not supported by this receive queue, insufficient buffering to receive a new RDMA or Atomic Operation request, or the length specified in an RDMA request is greater than 2^{31} bytes.

  In the case where the buffer size is insufficient to handle the request, the number of bytes transferred into the buffer is indeterminate. However, the CI shall not write beyond the buffer bounds.
- Remote Access Error - A protection error occurred on a remote data buffer to be read by an RDMA Read, written by an RDMA Write or accessed by an atomic operation. This error is reported only on RDMA operations or atomic operations.
- Remote Operation Error - The operation could not be completed successfully by the responder. Possible causes include a responder QP related error that prevented the responder from completing the request or a malformed WQE on the Receive Queue.
- Transport Retry Counter Exceeded - The local transport timeout retry counter was exceeded while trying to send this message.
- RNR Retry Counter Exceeded - The RNR NAK retry count was exceeded.

The following errors are reported only for RD QPs or EE Contexts.

- Remote Invalid RD Request - The responder detected an invalid incoming RD message. Causes include a Q_Key or RDD violation.
- Remote Aborted Error - The requester aborted the operation. One possible cause is the requester suspended the operation and will retry it later using a new Receive WQE. The other possible cause is the requester abandoned the operation and placed the requester QP in the SQEr state.
- Invalid EE Context Number - An invalid EE Context number was detected.
- Invalid EE Context State - Operation is not legal for the specified EE Context state.

11.6.3 ASYNCHRONOUS EVENTS

This section describes the asynchronous events. Asynchronous events are separated into three categories; Affiliated asynchronous events, Affiliated asynchronous errors and Unaffiliated asynchronous errors. Both kinds of asynchronous errors are defined in 10.10.2.3 Asynchronous Errors on page 462.

Affiliated asynchronous events have been separated into two categories because the behavior of the QP/EE Context when the events occur are different.

C11-34: When an affiliated asynchronous error occurs, the CI **shall** cause the QP/EE to transition to the Error state.

C11-35: When an affiliated asynchronous event occurs, the CI **must** leave the QP/EE in the QP/EE State that it was in when the asynchronous event occurred.

Unaffiliated asynchronous errors are those which cannot be associated with a specific QP or EE Context.

The Verbs Consumer must register a handler as described in Set Asynchronous Event Handler to be notified that an asynchronous event has occurred. This mechanism is used to collect information about both events and the errors.

11.6.3.1 AFFILIATED ASYNCHRONOUS EVENTS

Affiliated asynchronous events are advisories to the Verb Consumer that the specified event has occurred on the specified QP or EE Context. Events in this category are not considered to be errors by the Channel Interface, so the QP/EE state remains unchanged.

- Path Migrated - Indicates the connection has migrated to the alternate path.
- Communication Established - Indicates the first packet has arrived for the Receive Work Queue where the QP/EE is still in the RTR state. The handle of the QP/EE, which was the destination of this packet is returned in the event record. This event may be used by the Communication Manager as shown in the state diagram in 12.9.6 Communication Establishment - Passive on page 575 and described in CM Passive States. The Communication Manager may receive this event while it is already in the Established state; this is not an error.

C11-36: The CI **shall** generate a Communication Established asynchronous event when the first packet arrives for the Receive Work Queue when the QP/EE is still in the RTR state.

- Send Queue Drained - Indicates that the Send Queue of the specified Queue Pair has completed the outstanding messages in progress when the state change was requested and, if applicable, has received all acknowledgements for those messages.

11.6.3.2 AFFILIATED ASYNCHRONOUS ERRORS

- CQ Error - Indicates an error occurred when writing an entry to the Completion Queue.

C11-37: The CI **shall** generate a CQ Error when an error, other than CQ overrun, occurs while writing an entry to the CQ.

C11-38: The CI **shall** generate a CQ Error when a CQ overrun is detected.

This condition will result in an Affiliated Asynchronous Error for any associated Work Queues when they attempt to use that CQ. Completions can no longer be added to the CQ. It is not guaranteed that completions present in the CQ at the time the error occurred can be retrieved. Possible causes include a CQ overrun or a CQ protection error.

- Local Work Queue Catastrophic Error - An error occurred while accessing or processing the Work Queue that prevents reporting of completions.

C11-39: The CI **shall** generate a Local Work Queue Catastrophic Error when a Work Queue associated with a CQ that caused the CQ Error to be generated attempts to use that CQ.

C11-40: The CI **shall** generate a Local Work Queue Catastrophic Error when an error occurred while accessing or processing the Work Queue that prevents reporting of completions.

- Local EE Context Catastrophic Error - An Error occurred while accessing or processing the EE Context that prevents reporting of completions.

o11-5.a1: If the CI supports RD Service, the CI **shall** generate a EE Context Catastrophic Error when an error occurred while accessing or processing the EE Context that prevents reporting of completions.

- Path Migration Request Error - Indicates the incoming path migration request to this QP/EE was not accepted. The validation process is defined in section Migration Request.

o11-6: If the CI supports automatic path migration, the CI **shall** generate a Path Migration Request Error when the incoming path migration request to this QP/EE was not accepted.

11.6.3.3 UNAFFILIATED ASYNCHRONOUS ERRORS

- Local Catastrophic Error - An error occurred which cannot be attributable to any resource and CI behavior is indeterminate.

C11-41: The CI **shall** generate a Local Catastrophic Error when an error occurred which cannot be attributable to any resource and CI behavior is indeterminate.

- Port Error - Issued when the link is declared unavailable.

C11-42: The CI **shall** generate a Port Error when the link is declared unavailable.

Using the definitions of Link States, the "unavailable" states are considered to be: Down, Initialize and Armed. The "available" states are Active and ActDefer. The "Port Error" unaffiliated asynchronous error is generated when the link associated with an HCA port transitions from an available to an unavailable state.

CHAPTER 12: COMMUNICATION MANAGEMENT

12.1 OVERVIEW

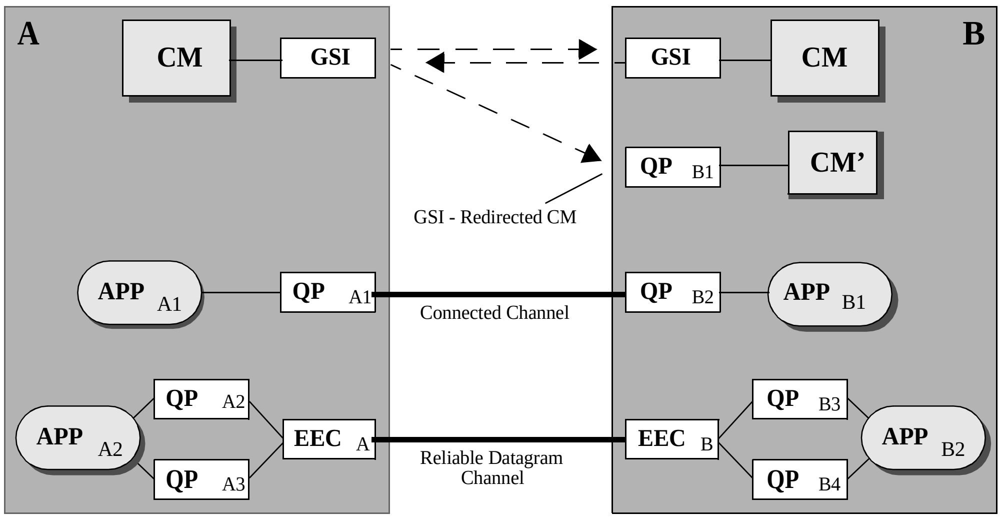


Figure 126 Communication Management Entities

Communication Management encompasses the protocols and mechanisms used to establish, maintain, and release channels for the IB Reliable Connection, Unreliable Connection, and Reliable Datagram transport service types. The Service ID Resolution Protocol (see section 12.11) enables users of Unreliable Datagram service to locate Queue Pairs supporting their desired service.

Connections are managed over Queue Pairs other than those used for the connection, through the protocol described herein, between the Communication Managers (CMs) on each system. (See Figure 126) The CMs communicate using Management Datagrams (MADs), typically over the General Services Interface (GSI) on each system. This document defines CM external behaviors, but internal interfaces and implementations are outside the scope of the InfiniBandTM Architecture specification. Examples are intended to enable understanding, not to specify implementation.

At creation, QPs and EECs are not ready for communication. The attributes of the QP/EEC must be modified (see sections 11.2.3.2 and 11.2.6.2) to support the desired communication characteristics and target(s).

Due to their nature, raw packet QPs do not need, and are not supported by, IB communication management.

The requirements on participating CMs are not equal. The initiating CM is responsible for collecting or calculating most of the information necessary to establish the connection. Much of the raw information is available from Subnet Administration, but some adjustments may be desirable, depending on the application of the channel.

CMs must maintain a certain amount of information for the lifetime of a connection. Details may be found in section 12.9.9 .

12.2 ESTABLISHMENT

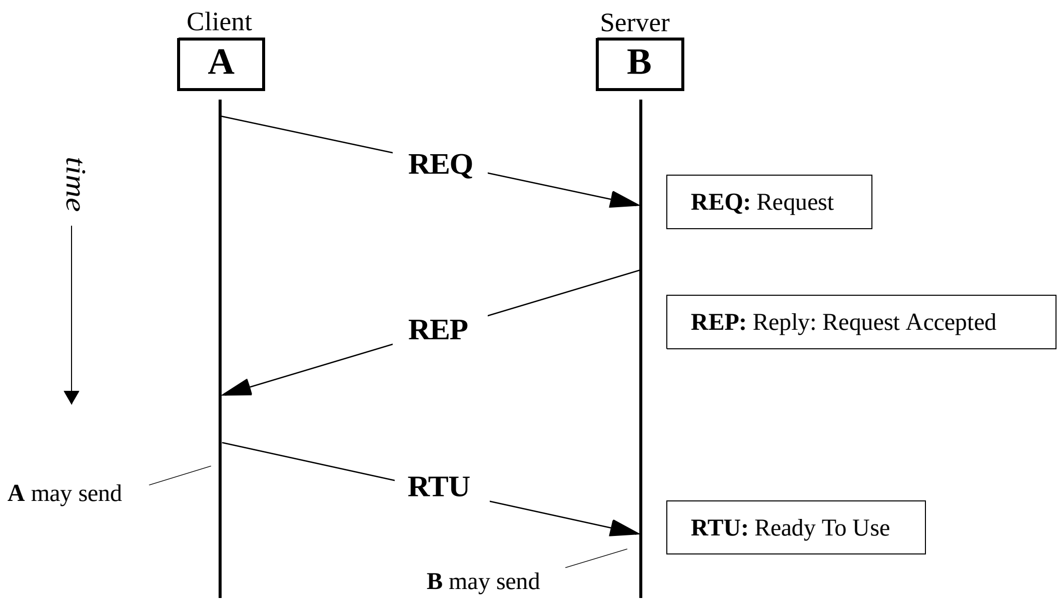


Figure 127 Sample Connection Establishment Sequence

Two models are supported by the Connection Establishment protocol: Active/Passive (Client/Server), and Active/Active (Peer to Peer).

As seen in Figure 126, the CMs on each system establish connections on behalf of their clients. The interactions between CMs and their clients are outside the scope of this specification.

In the Active/Passive model (shown in Figure 127), B's CM waits for connection requests on behalf of a server (e.g., a server process or an I/O controller) that waits (passive) for connections to be established by clients. The CM (A) for a prospective client (active) places the ServiceID that designates the desired service in the Request (**REQ**) message that begins the connection establishment sequence. The ServiceID allows the passive-side CM (B) to associate the request with the appropriate server entity. Should the **REQ** be accepted, B's CM returns the Queue Pair Number (QPN) (and End to End Context Number (EECN) for RD service) in a Response (**REP**) MAD. Whether QPs and EECs are pre-allocated or are allocated in response to a request is an implementation consideration that is outside the scope of the IBA specification.

In the Active/Active model, both entities begin as active (i.e., both send **REQ**), but one ultimately takes the passive role for establishing the connection. The selection of the passive entity is described in section 12.10.4 .

12.3 AUTOMATIC PATH MIGRATION

The connection establishment messages specify the information necessary to support an (optional) alternate pair of endpoints to support Automatic Path Migration (APM). APM is described in section 17.2.8.1 and the support mechanisms are described in section 10.4 . Channel Adapters that do not support APM may ignore the Alternate address information.

12.4 RELEASE

Connections are released through the exchange of Disconnect Request (**DREQ**) and Disconnect Reply (**DREP**) MADs. Communicating entities will likely wish to effect an orderly shutdown of their protocol before initiating the Disconnect sequence. After a connection is released, the CM shall cause each involved QP/EEC to be placed into the TimeWait state as defined in section 12.9.8.4 .

CMs shall maintain enough connection state information to detect an attempt to initiate a connection on a remote QP/EEC that has not been released from a connection with a local QP/EEC, or that is in the TimeWait state. Such an event could occur if the remote CM had dropped the connection and sent **DREQ**, but the **DREQ** was not received by the local CM. If the local CM receives a **REQ** that includes a QPN (or EECN if **REQ:RDC Exists** is not set), that it believes to be connected to a local QP/EEC, the local CM shall act as defined in section 12.9.8.3 .

12.4.1 STALE CONNECTION

A QP/EEC is said to have a stale connection when only one side has connection information. A stale connection may result if the remote CM had dropped the connection and sent a **DREQ** but the **DREQ** was never received by the local CM. Alternatively the remote CM may have lost all record of past connections because its node crashed and rebooted, while the local CM did not become aware of the remote node's reboot and therefore did not clean up stale connections.

12.5 SERVICE TYPES

12.5.1 SUPPORTED PROTOCOLS

The sections that follow contain message descriptions and state diagrams specifying how those messages are exchanged. The messages are used for the following purposes:

- To support connection establishment for RC and UC service types.
- To support end to end context establishment for RD service.

Figure 126 illustrates the following relationships.

12.5.2 CONNECTED SERVICES

A channel is established for Reliable Connected (RC) and Unreliable Connected (UC) service types by reaching agreement between the end CMs.

12.5.3 UNRELIABLE DATAGRAM SERVICE

Unreliable Datagram (UD) service allows a message to be sent to any destination, although there is no guarantee that the destination will receive or accept it. The ServiceID resolution facility (Section 12.11) may be used to determine the appropriate target QP.

12.5.4 RELIABLE DATAGRAM

Reliable Datagram (RD) service allows multiple Queue Pairs to communicate over a single RD channel (defined by a pair of EE contexts). One QP on each end is specified when an RD channel is established. A pair of applications using these QPs that wish to use additional QPs over that RDC do not need to use CM to associate those QPs. Application-specific messages could be sent over the original QPs to notify the other side of the QPNs of the new QPs.

Unless otherwise specified, an RD communication request implies the creation of a new RDC. Setting the **RDC Exists** field in the **REQ** message allows the reuse of the specified RDC. (See section 12.6.5)

12.6 COMMUNICATION MANAGEMENT MESSAGES

The following sections describe the set of messages used to support the communication establishment scenarios supported by the IBA:

a) Active client to passive server

b) Active client to active client

c) Active client to passive server (with third-party redirector)

12.6.1 REQUIRED MESSAGES

All IBA hosts and all IBA targets that support RC, UC, or RD service types shall support the following messages:

- Request for Communication (**REQ**) (Section 12.6.5)
- Message Receipt Acknowledgement (**MRA**) (Section 12.6.6) All IBA hosts and targets are required to be able to receive and act upon an **MRA**, but the ability to send an **MRA** is optional.
- Reject (**REJ**) (Section 12.6.7)
- Reply to Request for Communication (**REP**) (Section 12.6.8)
- Ready to Use (**RTU**) (Section 12.6.9)
- Request for Communication Release (**DREQ**) (Section 12.6.10)
- Reply to Request for Communication Release (**DREP**) (Section 12.6.11)

C12-1: A CA that supports Reliable Connected, Unreliable Connected, or Reliable Datagram channels **shall** support their establishment using the CM protocol.

C12-2: For the states and messages it supports, a CM **shall** adhere to the CM protocol as defined in sections 12.9.7 and 12.9.8 .

C12-3: CM message contents **shall** conform to the field descriptions in section 12.7 .

C12-4: A CM **shall** support sending the **REJ** message in accordance with section 12.6.7 .

C12-5: A CM **shall**, upon receipt of an **MRA** message, behave in accordance with section 12.9.8.5 .

o12-1: If a CM sends the **REQ** message, it **shall** do so in accordance with section 12.6.5 .

o12-2: If a CM sends the **MRA** message, it **shall** do so in accordance with section 12.6.6 .

o12-3: If a CM sends the **REP** message, it **shall** do so in accordance with section 12.6.8 .

o12-4: If a CM sends the **RTU** message, it **shall** do so in accordance with section 12.6.9 .

o12-5: If a CM sends the **DREQ** message, it **shall** do so in accordance with section 12.6.10 .

o12-6: If a CM sends the **DREP** message, it **shall** do so in accordance with section 12.6.11 .

o12-7: If a CM initiates connection requests (active role), it **shall** support sending the **REQ**, **RTU**, **DREQ**, and **DREP** messages, and responding to the **REP**, **DREQ**, and **DREP** messages.

o12-8: If a CM accepts connection requests (passive role), it **shall** support responding to the **REQ**, **RTU,** and **DREQ** messages, and sending the **REP** and **DREP** messages.

o12-9: If a CM sends the **DREQ** message, it **shall** be able to handle the **DREP** message.

12.6.2 CONDITIONALLY REQUIRED MESSAGES

Support for these messages is required if non-management services are provided on the Channel Adapter at other than fixed QPNs. Management services include those provided through Subnet Management Packets (see 14.2 Subnet Management Class) or through General Management Packets (see Chapter 16: General Services).

- Service ID Resolution Request (**SIDR_REQ**) (Section 12.11.1)
- Service ID Resolution Response (**SIDR_REP**) (Section 12.11.2)

o12-10: If a CM sends the **SIDR_REQ** message, it **must** do so in accordance with section 12.11.1 .

o12-11: If a CM sends the **SIDR_REP** message, it **must** do so in accordance with section 12.11.2 .

o12-12: If a CA provides services (other than Subnet Management and General Services) using the UD service type at other than fixed QPNs, its CM **must** support receiving, processing and replying to the **SIDR_REQ** message as specified in section 12.11 .

12.6.3 OPTIONAL MESSAGES

Support for these messages is optional:

- Load Alternate Path (**LAP**) (Section 12.8.1)
- Alternate Path Response (**APR**) (Section 12.8.2)

o12-13: If a CM accepts **REQ** messages and agrees to perform Automatic Path Migration, it **shall** support receiving, processing and replying to the **LAP** message as specified in section 12.8 .

o12-14: If a CM sends **REQ** messages with Alternate Port/Path information, it **shall** support sending the **LAP** message as specified in section 12.8 .

12.6.4 Message Usage

Connected Transport Service Types require state information to be established, maintained, and released at both ends of the connection. Consumers can use the messages described in this section for that purpose.

By definition, unreliable datagram communications do not require any connection state to be established, maintained, or released. However, communication services are provided to allow local and remote QPs to be associated based on a specific Service ID. (See section 12.11)

Reliable datagram communication requires Reliable Datagram Channels to be created, maintained, and released between CAs.

The Communication Management information contained in each Management Datagram message is described below. The MAD header format is defined in 16.7.1 MAD Format on page 818.

The messages defined below are used for both establishing connections and end to end context establishment. The message definitions are the union of the fields required for both of these purposes, and therefore there are some fields in the messages which are useful for connection establishment but not for end to end context establishment, and vice versa. This is done to decrease the total number of message types in the protocol. For each field in a message, whether the field is intended to support connection establishment or end to end context establishment (or both) is noted.

12.6.5 REQ - Request for Communication

REQ is sent to initiate the communication establishment sequence. The initiator (**REQ** sender) provides the Port Address (GID and/or LID) and the Queue Pair Number that it will be using for its end of the channel. For Reliable Datagram Channel establishment, the EE Context Number is included.

The initiator is responsible for proposing the Port Addresses (Primary and optional Alternate) that the target (**REQ** recipient) is to use for the channel. Based on the path defined by those port addresses, the initiator provides timeout information and the Service Level to be used by the target for any messages that it initiates. The SL from initiator to target need not be the same as from target to initiator, but the same SL must be used for both the request packets and any associated ACK or NAK packets associated with that request. Path information is available from Subnet Administration (see section 15.2.5.17 PathRecord).

For service resolution and QP association over already existing Reliable Datagram Channels, **REQ:RDC Exists** must be set.

Table 86 REQ Message Contents

Field	Description	Used for Purpose	Byte [Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1 .	C, EE	0	32	
(reserved)			4	32	
ServiceID	See section 12.7.3 .	C, EE	8	64	
Local CA GUID	See section 12.7.9	C, EE	16	64	
Local CM Q_Key	See section 12.7.8	C, EE	24	32	
Local Q_Key	See section 12.7.13	EE	28	32	
Local QPN	See section 12.7.12 .	C, EE	32	24	
Offered Responder Resources	See section 12.7.29	C, EE	35	8	
Local EECN	See section 12.7.14	EE	36	24	
Offered Initiator Depth	See section 12.7.30	C, EE	39	8	
Remote EECN	See section 12.7.15	EE	40	24	
Remote CM Response Timeout	See section 12.7.4	C, EE	43	5	
Transport Service Type	See section 12.7.6 .	C, EE	43 [5]	2	
End-to-End Flow Control	See section 12.7.26	C, EE	43 [7]	1	
Starting PSN	See section 12.7.31	C, EE	44	24	
Local CM Response Timeout	See section 12.7.5	C, EE	47	5	
Retry Count	See section 12.7.38	C, EE	47[5]	3	
Partition Key	See section 12.7.24	C, EE	48	16	
Path Packet Payload MTU	See section 12.7.28	C, EE	50	4	
RDC Exists	Whether RDC already exists.	EE	50[4]	1	1 if RDC exists, 0 if RDC does not
RNR Retry Count	See section 12.7.39	C, EE	50[5]	3	
Max CM Retries	See section 12.7.27	C, EE	51	4	
(reserved)			51[4]	4	
Primary Local Port LID	See section 12.7.11 .	C, EE	52	16	
Primary Remote Port LID	See section 12.7.21 .	C, EE	54	16	
Primary Local Port GID	See section 12.7.10 .	C, EE	56	128	
Primary Remote Port GID	See section 12.7.20 .	C, EE	72	128	

Table 86 REQ Message Contents

Field	Description	Used for Purpose	Byte [Bit] Offset	Length, bits	Values
Primary Flow Label	See section 12.7.18	C, EE	88	20	
(reserved)			90[4]	4	
Primary Packet Rate (IPD)	See section 12.7.25	C, EE	91	8	
Primary Traffic Class	See section 12.7.17	C, EE	92	8	
Primary Hop Limit	See section 12.7.19	C, EE	93	8	
Primary SL	See section 12.7.16	C, EE	94	4	
Primary Subnet Local	See section 12.7.7	C, EE	94 [4]	1	
(reserved)			94 [5]	3	
Primary Local ACK Timeout	See section 12.7.34	C, EE	95	5	
(reserved)			95[5]	3	
Alternate Local Port LID	See section 12.7.11	C, EE	96	16	
Alternate Remote Port LID	See section 12.7.23 .	C, EE	98	16	
Alternate Local Port GID	See section 12.7.10 .	C, EE	100	128	
Alternate Remote Port GID	See section 12.7.22 .	C, EE	116	128	
Alternate Flow Label	See section 12.7.18	C, EE	132	20	
(reserved)			134[4]	4	
Alternate Packet Rate (IPD)	See section 12.7.25	C, EE	135	8	
Alternate Traffic Class	See section 12.7.17	C, EE	136	8	
Alternate Hop Limit	See section 12.7.19	C, EE	137	8	
Alternate SL	See section 12.7.16	C, EE	138	4	
Alternate Subnet Local	See section 12.7.7	C, EE	138[4]	1	
(reserved)			138[5]	3	
Alternate Local ACK Timeout	See section 12.7.34	C, EE	139	5	
(reserved)			139[5]	3	
PrivateData	See section 12.7.35	C, EE	140	736	

12.6.6 MRA - MESSAGE RECEIPT ACKNOWLEDGMENT

MRA is sent in response to a **REQ** or **REP** message when the recipient of the message anticipates that it will not be able to respond within the time specified by **REQ**:Remote CM Response Timeout . **MRA** is sent to prevent the other party in the communication establishment protocol from either unnecessarily timing out the communication establishment attempt or flooding the link with unnecessary retries.

Table 87 MRA Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1 .	C, EE	0	32	
Remote CommunicationID	See section 12.7.2 .	C, EE	4	32	
Message MRAed	The message being MRAed.	C, EE	8	2	0x0 - **REQ**, 0x1 - **REP** 0x2 - **LAP**
(reserved)			8[2]	6	
ServiceTimeout	See section 12.7.32	C, EE	9	5	
(reserved)			9[5]	3	
PrivateData	See section 12.7.35 .	C, EE	10	1776	

12.6.7 REJ - REJECT

REJ indicates that the sender will not continue through the communication establishment sequence, and the reason why it will not.

Table 88 REJ Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1 .	C, EE	0	32	0 if REJecting a **REQ** and no **MRA** was sent
Remote CommunicationID	See section 12.7.2 .	C, EE	4	32	
Message REJected	The message being REJected.	C,EE	8	2	0x0 - **REQ** 0x1 - **REP** 0x2 - Unknown/No message
(reserved)			8[2]	6	
Reject Info Length	If non-zero, the length in bytes of valid Additional Reject Information	C, EE	9	7	
(reserved)			9[7]	1	

Table 88 REJ Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Reason	Error code indicating the reason for the sender's termination of the communication establishment process.	C, EE	10	16	
Additional Reject Information (ARI)		C, EE	12	576	
PrivateData	See section 12.7.35 .	C, EE	84	1184	

12.6.7.1 EXAMPLE REJ MESSAGE

The content of the fields of a **REJ** that rejects a **REQ** because of an unacceptable primary port LID and suggests that a primary port LID of 200 be used are shown in the table below.

Table 89 Example REJ Message

Field	Contents
Local Communication ID	0
Remote Communication ID	0
Message REJected	0
Reason	15
Reject Info Length	2
Additional Reject Information (ARI)	200
PrivateData	empty

12.6.7.2 REJECTION REASON

Code	Reason	Description	Meaning of Additional Reject Information Field
1	No QP available	The **REQ** message required the recipient to allocate a QP, and none were available	
2	No EEC available	The **REQ** message required the recipient to allocate an EE context, and none were available	

Code	Reason	Description	Meaning of Additional Reject Information Field
3	No resources available	The **REQ** message required the recipient to allocate resources other than QPs or EE contexts, and none were available	
4	Timeout	The CM protocol timed out waiting for a message	
5	Unsupported request	Receiving CM does not support this request.	
6	Invalid Communication ID	The recipient received a CM message in which the Local Communication ID, Remote Communication ID, or both, were invalid.	
7	Invalid Communication Instance	The Local Communication ID, Remote Communication ID, QPN/EECN tuple does not refer to any valid communication instance.	
8	Invalid Service ID	The recipient of the **REQ** message does not recognize or does not support the service associated with the specified ServiceID	
9	Invalid Transport Service Type	The recipient of the **REQ** message did not recognize the requested Transport Service Type	
10	Stale connection	The recipient of the **REQ** determined that it already had a connection with the "Local QPN" or "Local EECN" specified in the **REQ**. Upon receiving a **REJ** with this reason, the **REJ** recipient shall cause the QP or EE context to be placed into the TimeWait state as described in section 12.9.8.4 .	
11	RDC does not exist	The Reliable Datagram Channel described in the **REQ** (Local EECN/Remote EECN) does not exist.	
12	Primary Remote Port GID rejected	The recipient of the **REQ** message could not (or would not) accept the Primary Remote Port GID	GID of acceptable port.
13	Primary Remote Port LID rejected	The recipient of the **REQ** message could not (or would not) accept the Primary Remote Port LID	LID of acceptable port.
14	Invalid Primary SL	The recipient of the **REQ** message does not support the requested Primary SL	Acceptable SL.
15	Invalid Primary Traffic Class	The recipient of the **REQ** message does not support the requested Primary Traffic Class	Acceptable Traffic Class
16	Invalid Primary Hop Limit	The recipient of the **REQ** message could not (or would not) accept the Primary Hop Limit	Acceptable Hop Limit
17	Invalid Primary Packet Rate	The recipient of the **REQ** message could not adjust its transmitter to send as slowly as would be required to comply with the requested Primary Packet Rate	Minimum acceptable Packet Rate
18	Alternate Remote Port GID rejected	The recipient of the **REQ** message could not (or would not) accept the Alternate Remote Port GID	GID of acceptable port.

Code	Reason	Description	Meaning of Additional Reject Information Field
19	Alternate Remote Port LID rejected	The recipient of the **REQ** message could not (or would not) accept the Alternate Remote Port LID	LID of acceptable port.
20	Invalid Alternate SL	The recipient of the **REQ** message does not support the requested Alternate SL	Acceptable SL.
21	Invalid Alternate Traffic Class	The recipient of the **REQ** message does not support the requested Alternate Traffic Class	Acceptable Traffic Class
22	Invalid Alternate Hop Limit	The recipient of the **REQ** message could not (or would not) accept the Altermate Hop Limit	Acceptable Hop Limit
23	Invalid Alternate Packet Rate	The recipient of the **REQ** message could not adjust its transmitter to send as slowly as would be required to comply with the requested Alternate Packet Rate	Minimum acceptable Packet Rate
24	Port and CM Redirection	The recipient of the **REQ** message supports the requested Service ID, but at the endpoint specified by the ARI. Further CM messages should be sent to that endpoint as well.	A ClassPortInfo data structure as documented in Section 13.4.8.1 describing where to send subsequent CM messages, and also describing the GID of the port to propose in the new **REQ**.
25	Port Redirection	The recipient of the **REQ** message supports the requested Service ID, but at the port specified by the ARI. Further CM messages shall be sent to the port to which the original **REQ** was sent.	GID of port to propose in new **REQ**.
26	Invalid Path MTU	The recipient of the **REQ** message cannot support the maximum packet payload size specified	Maximum acceptable maximum packet payload size
27	Insufficient Responder Resources	The value of Responder Resources (for RDMA Read/Atomics) in the **REP** message was insufficient.	
28	Consumer Reject	The consumer decided to reject the communication or EE context setup establishment attempt for reasons other than those listed above. (Typically this happens based upon information being conveyed in the PrivateData field of a message.)	Defined by the consumer
29	RNR Retry Count Reject	The recipient of the message rejects the RNR NAK Retry count value.	

12.6.8 REP - REPLY TO REQUEST FOR COMMUNICATION

REP is returned in response to **REQ**, indicating that the respondent accepts the ServiceID, proposed primary port, and any parameters specified in the PrivateData area of the **REQ**.

Table 90 REP Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits	Values
Local Communication ID	See section 12.7.1 .	C, EE	0	32	
Remote Communication ID	See section 12.7.2 .	C, EE	4	32	Value present in **REQ**
Local Q_Key	See section 12.7.13	EE	8	32	
Local QPN	See section 12.7.12 .	C, EE	12	24	
(reserved)			15	8	
Local EE Context Number	See section 12.7.14	EE	16	24	
(reserved)			19	8	
Starting PSN	See section 12.7.31	C, EE	20	24	
(reserved)			23	8	
Responder Resources	See section 12.7.29	C, EE	24	8	
Initiator Depth	See section 12.7.30	C,EE	25	8	
Target ACK Delay	See section 12.7.33	C, EE	26	5	
Failover Accepted	See section 12.7.36 .	C, EE	26[5]	2	0: Failover accepted 1: Failover port rejected because failover is not supported. Alternate Path parameters were not checked. 2: Failover is supported and all Alternate Path parameters are valid, but the failover port was rejected for some other reason.
End-To-End Flow Control	See section 12.7.26	C, EE	26[7]	1	
RNR Retry Count	See section 12.7.39	C,EE	27	3	
(reserved)			27[3]	5	
PrivateData	See section 12.7.35	C, EE	28	1632	

12.6.9 RTU - READY TO USE

RTU indicates that the connection is established, and that the recipient may begin transmitting.

Table 91 RTU Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, Bits
Local Communication ID	See section 12.7.1 .	C, EE	0	32
Remote CommunicationID	See section 12.7.2 .	C, EE	4	32
PrivateData	See section 12.7.35	C, EE	8	1792

12.6.10 DREQ - REQUEST FOR COMMUNICATION RELEASE (DISCONNECTION REQUEST)

DREQ is sent to initiate the connection release sequence.

Table 92 DREQ Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1 .	C, EE	0	32
Remote CommunicationID	See section 12.7.2 .	C, EE	4	32
Remote QPN/EECN	See section 12.7.37	C, EE	8	24
(reserved)			11	8
PrivateData	See section 12.7.35	C, EE	12	1760

The values for Local and Remote Communication ID are those that were used to create the channel.

12.6.11 DREP - REPLY TO REQUEST FOR COMMUNICATION RELEASE

DREP is sent in response to **DREQ**, and signifies that the sender has received the **DREP**.

Table 93 DREP Message Contents

Field	Description	Used for Purpose	Byte[Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1 .	C, EE	0	32
Remote CommunicationID	See section 12.7.2 .	C, EE	4	32
PrivateData	See section 12.7.35	C, EE	8	1792

12.7 MESSAGE FIELD DETAILS

The following table summarizes each of the message fields, and indicates where the consumer can find the contents necessary to populate the field.

Table 94 Message Field Origins

Field	Populated From
Local Communication ID	The consumer sending the **REQ** message chooses this value. See section 12.7.1
Remote CommunicationID	The consumer replying to the **REQ** message chooses this value. See section 12.7.2
Service ID	Assuming that the consumer uses the InfiniBand™ service naming facility, this comes from the ServiceRecord, as defined in section 15.2.5.15 ServiceRecord.
Remote CM Response Timeout	The consumer should set this field to be large enough to allow enough time under normal circumstances for the recipient to be able to process the incoming message and have the response message traverse the path between source and destination. The service time at the recipient depends upon the service being requested, but the maximum time it could take to successfully traverse the path can be found in the PathRecord as defined in section 15.2.5.17 PathRecord. (How the particular path to be used is selected is a policy decision that is left up to the consumer.)
Local CM Response Timeout	This timeout period needs to allow for the path between the source and destination to be traversed twice, and also to allow for the **REP** message to be processed. The amount of time it takes to service the **REP** message may depend upon the service that was requested, but the maximum time it could take to successfully traverse the path can be found in the PathRecord as defined in section 15.2.5.17 PathRecord.
Transport Service Type	The consumer sets this based upon the type of service it is requesting: Reliable Connected, Unreliable Connected, or Reliable Datagram.
Subnet Local	This can be determined by comparing the PortInfo:SubnetPrefix fields associated with the Local Port GID and the Remote Port GID. The PortInfo record is defined in section 15.2.5.3 PortInfoRecord.
Local CM Q_Key	The consumer can determine this for an HCA by querying the Queue Pair that it is using to send the message. The Query Queue Pair verb is defined in section 11.2.3.3 Query Queue Pair. (How this information is determined for a TCA is implementation-specific.)

Table 94 Message Field Origins

Field	Populated From
Local CA GUID	This information can be found in the NodeInfo:NodeGUID field, as defined in section 14.2.5.3 NodeInfo. (Which CA to use is a policy decision that is left up to the consumer.)
Local Port GID	This information can be found in the GidInfo record, as defined in section 15.2.5.20 GuidInfoRecord. (Which port on the CA to use and which of the available GIDs on the chosen port to use is a policy decision that is left up to the consumer.)
Local Port LID	This information can be found in the PortInfo:LID field, as defined in section 14.2.5.6 PortInfo. (Which port on the CA to use and which of the available LIDs on the chosen port to use is a policy decision that is left up to the consumer.)
Local QPN	The consumer can determine this for an HCA by querying the Queue Pair that it is offering up for connection establishment. The Query Queue Pair verb is defined in section 11.2.3.3 Query Queue Pair. (How this information is determined for a TCA is implementation-specific.)
Local Q_Key	The consumer can determine this for an HCA by querying the Queue Pair that it is offering up for connection establishment. The Query Queue Pair verb is defined in section 11.2.3.3 Query Queue Pair. (How this information is determined for a TCA is implementation-specific.)
Local EECN	The consumer can determine this for an HCA by querying the EE Context that it is offering up for communications establishment. The Query EE Context verb is defined in section 11.2.6.3 Query EE Context. (How this information is determined for a TCA is implementation-specific.)
Remote EECN	The data originates on the remote end of an existing connection, and is returned to the local end in a **REP** message. It is determined by the remote end in the same manner as the Local EECN.
Service Level	This information can be found in the PathRecord:SL field, as defined in section 15.2.5.17 PathRecord.
Traffic Class	This information can be found in the PathRecord:TClass field, as defined in section 15.2.5.17 PathRecord.
Flow Label	The purpose of this field is to identify a group of packets that must be delivered in order. See section 8.3 Global Route Header for a description of how this value is chosen.
Hop Limit	This information can be found in the PathRecord:HopLimit field, as defined in section 15.2.5.17 PathRecord.
Primary Remote Port GID	This information can be found in the GidInfo record associated with the remote port, as defined in section 15.2.5.20 GuidInfoRecord. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.15 ServiceRecord.
Primary Remote Port LID	This information can be found in the PortInfo:LID field associated with the remote port, as defined in section 14.2.5.6 PortInfo. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.15 ServiceRecord.

Table 94 Message Field Origins

Field	Populated From
Alternate Remote Port GID	This information can be found in the GidInfo record associated with the remote port, as defined in section 15.2.5.20 GuidInfoRecord. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.15 ServiceRecord.
Alternate Remote Port LID	This information can be found in the PortInfo:LID field associated with the remote port, as defined in section 14.2.5.6 PortInfo. The port that should be targeted based on the service being requested can be found in the ServiceRecord, as defined in section 15.2.5.15 ServiceRecord.
Partition Key	This information can be found in the PathRecord:P_Key field, as defined in section 15.2.5.17 PathRecord.
Packet Rate	This information can be found in the PathRecord:Rate field, as defined in section 15.2.5.17 PathRecord.
End-to-End Flow Control	All HCAs are required to support End-to-End Flow Control, and so if the CA that the initiator is using is an HCA this field should be set to 1. Whether or not End-to-End Flow Control is supported by a TCA is an implementation option, and it is therefore outside the scope of the InfiniBand™ architecture to specify the origin of this field in a TCA.
Max CM Retries	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.27 .
Path Packet Payload MTU	This information can be found in the PathRecord:Mtu field, as defined in section 15.2.5.17 PathRecord.
Responder Resources	The consumer can determine the maximum supported value for a QP/EEC by querying the HCA that will be used for communication. The Query HCA verb is defined in section 11.2.1.2 Query HCA
Initiator Depth	The consumer can determine the maximum supporte value for a QP/EEC by querying the HCA that will be used for communication. The Query HCA verb is defined in section 11.2.1.2 Query HCA
Starting PSN	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.31 .
Service Timeout	The consumer should set this field to be large enough to allow enough time for it to complete the processing of the incoming message and have the response message that it sends out traverse the path between source and destination. The incoming message processing time depends upon the service being requested and potentially other state, but the maximum time it could take to successfully traverse the path can be found in the PathRecord as defined in section 15.2.5.17 PathRecord.
Target ACK Delay	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.33 Target ACK Delay.
Local ACK Timeout	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.34 Local ACK Timeout.
PrivateData	The contents of this field are outside the scope of what the InfiniBand™ specification defines; the usage (if any) of this field is specified by higher-level communications establishment protocols.
Failover Accepted	Set as per the description in section 12.7.36 Failover Accepted.

Table 94 Message Field Origins

Field	Populated From
Remote QPN/EECN	This should be the same as the Local QPN/Local EECN returned in the **REP** message.
Retry Count	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.38 Retry Count.
RNR Retry Count	The value of this field is a policy decision that is outside the scope of Communication Management to define. The field is discussed in section 12.7.39 RNR Retry Count.

12.7.1 Local Communication ID

An identifier that uniquely identifies this connection from the sender's point of view. The sender must use the same identifier for all phases of communication establishment and release. It must not reuse a Local Communication ID for the life of the connection, or while any messages related to the connection could still be in the fabric. (How long a message related to the connection could still be in the fabric is touched upon in section 12.9.8.4 .) The Communication ID allows the recipient to determine whether the message is a duplicate of an old message, or represents a new connection request.

12.7.2 Remote Communication ID

An identifier that uniquely identifies this connection from the recipient's point of view. (As an example, for a **REP** message this would be the same as the Local Communication ID that was received in the **REQ** message.) The values in the Local and Remote Communication ID fields in the Communication Management MADs are exchanged between requests and replies.

The pair of (Local Communication ID, Remote Communication ID) is used to reference connections during establishment, failover management, and release. CM messages with invalid Communication IDs shall not be processed, and shall be rejected as specified in section 12.6.7 .

12.7.3 ServiceID

An identifier that specifies the service being requested. The ServiceID field specifies the service number desired by the requestor. These include, but are not limited to, the service numbers defined for typical TCP services. The mappings between services and ServiceIDs are outside the scope of Communication Management.

12.7.4 Remote CM Response Timeout

The time, expressed as ($4.096\ \mu S * 2^{\text{Remote CM Response Timeout}}$), within which the CM message recipient shall transmit a response to the sender.

This value is unsigned. The recipient uses this information to determine whether it should send an **MRA**. (See section 12.9.8.5)

12.7.5 LOCAL CM RESPONSE TIMEOUT

The time, expressed as ($4.096\ \mu S * 2^{\text{Local CM Response Timeout}}$), that the remote CM shall wait for a response from the local CM to a CM message sent by the remote CM. This value is unsigned. Note that whereas Remote CM Response Timeout is the time between receipt of a message and transmission of a response, Local CM Response Timeout includes that "turn-around" time, as well as round trip packet flight time. (See section 12.9.8.5) The initiating CM is responsible for determining this value, through Subnet Management or other means.

12.7.6 TRANSPORT SERVICE TYPE

Specifies desired service type: Reliable Connected, Unreliable Connected, or Reliable Datagram. The field is encoded as follows:

0: RC
1: UC
2: RD
3: Reserved

12.7.7 SUBNET LOCAL

0: Local and remote are on different subnets (LID fields not valid)
1: Local and remote are on same subnet (GID fields are still valid, though)

12.7.8 LOCAL CM Q_KEY

The Q_Key used by the sending CM. This value must be used in messages sent to the CM.

12.7.9 LOCAL CA GUID

The EUI-64 GUID of the sending Channel Adapter.

12.7.10 LOCAL PORT GID

The GID of the local CA port on which the channel is to be established. If an alternate path is not to be specified, the **Alternate Local Port GID** field shall be set to zero. If this field is non-zero, it shall contain a valid GID.

12.7.11 LOCAL PORT LID

The LID of the local CA port on which the channel is to be established. If an alternate path is not to be specified, the **Alternate Local Port LID** field shall be set to zero.

12.7.12 LOCAL QPN

The QPN of the message sender's QP on which the channel is to be established. One Reliable Datagram QP may be associated with multiple EE contexts. A QPN must be specified when establishing an RD channel, but use of this QPN is not limited to this RDC. Once a consumer establishes a Reliable Datagram Channel, the consumer may use additional QPs over the RDC without an additional connection establishment exchange.

CM shall not be used to connect the Send Work Queue of a QP to the Receive Work Queue of the same QP. (If so desired, the consumer can do this using the Modify QP verb.) Attempting to do this may result in unpredictable behavior when doing connection establishment between peers.

12.7.13 LOCAL Q_KEY

(RD Only) The Q_Key for the QP specified by **Local QPN**.

12.7.14 LOCAL EECN

The EE Context Number for the message sender's end of the RD channel.

12.7.15 REMOTE EECN

The EE Context Number for the remote end of the existing Reliable Datagram channel. 0 if **REQ:RDC Exists** is not set.

12.7.16 SERVICE LEVEL

The value to be placed in the Service Level field for packets sent by the recipient. For more information on Service Levels, see section 7.6.5 Service Level on page 158.

12.7.17 TRAFFIC CLASS

Defines Traffic Class for globally-routed packets.

12.7.18 FLOW LABEL

Defines Flow Label for globally-routed packets.

12.7.19 HOP LIMIT

The maximum number of hops a packet can make between subnets before being discarded.

12.7.20 PRIMARY REMOTE PORT GID

The GID of the remote node's CA port on which the local node wishes to establish the channel. The remote node may send **REJ** to reject this port, and may optionally suggest an acceptable port.

12.7.21 Primary Remote Port LID

The LID of the remote node's CA port on which the local node wishes to establish the channel. The remote node may send **REJ** to reject this port, and may optionally suggest an acceptable port. The sender is responsible for ensuring that the LID and GID refer to the same port.

12.7.22 Alternate Remote Port GID

As in section 12.7.20 . A CA that does not support automatic failover shall set the **REP** 'Failover Accepted' field to one. If this field is zero, it shall contain a valid GID.

12.7.23 Alternate Remote Port LID

As in section 12.7.21 . A CA that does not support automatic failover shall set the **REP** 'Failover Accepted' field to one.

12.7.24 Partition Key

The Partition Key to be used for the channel being established.

12.7.25 Packet Rate

The maximum rate at which the remote may transmit over this channel, specified as described in section 9.11 Static Rate Control.

12.7.26 End-to-End Flow Control

Signifies whether the local CA actually implements End-to-End Flow Control (1), or instead always advertises 'infinite credits'(0). See section 9.7.7.2 End-to-End (Message Level) Flow Control for more detail.

12.7.27 Max CM Retries

Maximum number of times that either party can re-send a **REQ**, **REP**, or **DREQ** message. After re-sending for the maximum number of times without a response, the sending party should then terminate the protocol by sending a **REJ** message indicating that it timed out.

12.7.28 Path Packet Payload MTU

Specifies the maximum packet payload size, in bytes, for the channel being established. One of 256, 512, 1024, 2048, 4096. This value applies to both the primary and alternate paths.

12.7.29 Responder Resources

The maximum number of outstanding RDMA Read/Atomic operations the sender will support from the remote QP/EEC. This value may be zero. The maximum number that the HCA can support for a QP/EEC can be determined using the Query HCA verb. See section 11.2.1.2 Query HCA. Upon receiving the **REP** message, the requestor must decide whether the offered resources are sufficient for the intended use. If not, it may send the **REJ** message to discontinue the connection establishment.

12.7.30 INITIATOR DEPTH

The maximum number of outstanding RDMA Read/Atomic operations the sender will have to the remote QP/EEC. The Initiator Depth chosen by one side of the channel shall not exceed the Responder Resources offered by the other side. The maximum number that the HCA can support for a QP/EEC can be determined using the Query HCA verb. See section 11.2.1.2 Query HCA.

12.7.31 STARTING PSN

The transport Packet Sequence Number at which the remote node (relative to the sender of the **REQ** or **REP** message) shall begin transmitting over the newly established channel. This value should be chosen to minimize the chance that a packet from a previous connection could fall within the valid PSN window.

12.7.32 SERVICE TIMEOUT

Present in the **MRA**. The maximum time required for the sender to send a **REP**, **RTU**, **APR**, or **REJ** (as appropriate). This value is expressed as ($4.096\ \mu S*2^{\text{Service Timeout}}$) from the time the **MRA** is posted to the Send queue. The recipient of the MRA shall wait the specified time, plus a packet lifetime, after receiving this message before timing out. (See section 12.9.8.5) This value is unsigned.

12.7.33 TARGET ACK DELAY

($4.096\ \mu S*2^{\text{Target ACK Delay}}$) represents the maximum expected time interval between the target CA's reception of a message and the transmission of the associated ACK or NAK. This is information furnished by the target to the recipient. It provides the recipient with information about the maximum message processing latency of the target, which is one component of the overall time it takes to get an ACK or NAK after having sent a request packet. (The other component is the network propagation delay, which depends upon the configuration of the switches and routers between the two endpoints as well as the congestion in the network.) The recipient of the message containing the Target ACK Delay should use this value along with the recipient's best estimate of the network propagation delay to determine how long to wait before timing out a packet transmission to the target. This value is unsigned.

12.7.34 LOCAL ACK TIMEOUT

Value representing the transport (ACK) timeout for use by the remote, expressed as ($4.096\ \mu S*2^{\text{Local ACK Timeout}}$). Calculated by **REQ** sender, based on (2 * PacketLifeTime + Local CA's ACK delay). Although the remote CA is not required to use this value for its ACK timeout, it is strongly encouraged to do so. PacketLifeTime represents the maximum expected time interval consumed by a packet traversing the path between source and destination CA. PacketLifeTime is contained in the PathRecord, as defined in section 15.2.5.17 PathRecord. Local ACK Timeout is unsigned.

If too small a value is chosen for the Local ACK Timeout, the number of packet transmission timeouts reported by the remote CA may increase, which may increase the amount of work that is required in the CA to successfully send a packet. If too large a value is chosen, the amount of time that it takes to notice that a packet has not been successfully transmitted (e.g. due to a CRC error on the wire) will be increased, which may increase the amount of time it takes to recover from or report such errors.

12.7.35 PrivateData

Data that is opaque to the communication management protocol, passed from the sender to the recipient. The recipient may choose to accept or reject the request based on the private data. The format and meaning of the PrivateData field is specific to the ServiceID and message type, and is not specified within Communication Management.

12.7.36 Failover Accepted

Indicates whether the target of the **REQ** accepted or rejected the Alternate port address contained in the **REQ**. By sending the **REP**, the target accepts the connection request, but it may still reject the proposed failover port.

If failover is accepted, each CM shall cause the associated QP (for RC/UC) or EEC (for RD) specified by Local QPN to be placed in the REARM Migration State (see section 17.2.8.1 Automatic Path Migration Protocol).

If failover is rejected, each CM shall cause the associated QP or EEC to be placed in the Migstate:MIGRATED state upon transition to the RTR state.

12.7.37 Remote QPN/EECN

The remote (relative to the sender) QPN or EECN, as appropriate, that is the subject of the message. Provides an additional check that the (Local Communication ID, Remote Communication ID) pair references the correct resource.

12.7.38 Retry Count

The total number of times that the sender wishes the receiver to retry timeout, packet sequence, etc. errors before posting a completion error. See sections 9.9.2.1.1 Requester Error Retry Counters and 9.9.2.4.1 Requester Class A Fault Behavior for details of how the retry counter works.

12.7.39 RNR RETRY COUNT

The total number of times that the **REQ** or **REP** sender wishes the receiver to retry RNR NAK errors before posting a completion error. See sections 9.9.2.1.1 Requester Error Retry Counters and 9.9.2.4.1 Requester Class A Fault Behavior for details of how the RNR retry counter works.

12.8 ALTERNATE PATH MANAGEMENT

IBA supports Automatic Path Migration (see section 17.2.8 Automatic Path Migration), in which a channel's traffic (RC, UC, RD) may be moved to a pre-determined alternate path. The initial alternate path is established at connection setup, but if a migration occurs, a new path needs to be specified before re-enabling migration.

Two messages are specific to alternate path management. **LAP - Load Alternate Path** carries the new path information. **APR - Alternate Path Response** informs the requester of the status of the **LAP** request.

The **MRA** message may be sent by the **LAP** recipient if it is unable to send the **APR** message within the Remote CM Response Timeout. As the **LAP** is idempotent, the message may re-sent if there is no response, or if the Service Timeout is not met. The recipient **shall** return a failure status in the **APR** if the **LAP** request specifies an alternate path that is the same, in every respect, as the primary path. There is no limit on the number of **LAP** messages that a sender may have outstanding, but a sender shall have no more than one **LAP** outstanding per remote QP/EEC at any time.

The QP/EEC state changes requested by the **LAP** and **APR** messages may be effected through the ModifyQP or ModifyEE verbs (sections 11.2.3.2 and 11.2.6.2).

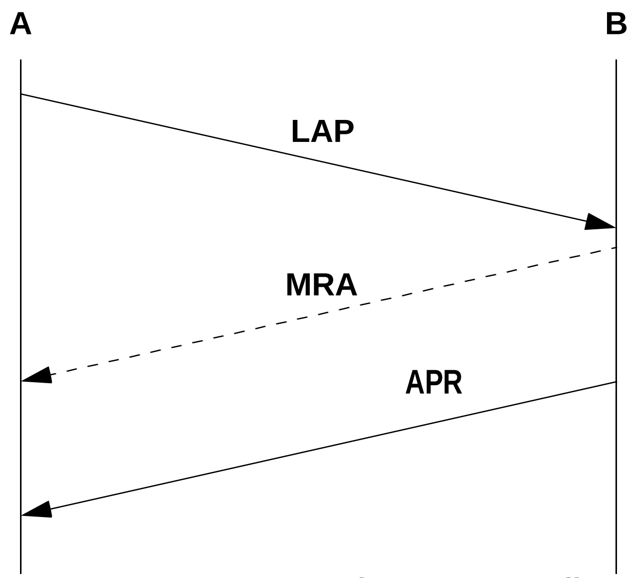


Figure 128 Loading alternate path

12.8.1 LAP - LOAD ALTERNATE PATH

LAP is an optional message used to change the alternate path information for a specific connection. It may be sent to update the alternate path information if fabric changes cause it to become invalid, or to load the "new" alternate path information after a path migration occurs. Loading alternate path information does not initiate the migration process for automatic failover; it just specifies which path is to be used when the path migration occurs.

Table 95 LAP Message Contents

Field	Description	Byte [Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1 .	0	32
Remote Communication ID	See section 12.7.2 .	4	32
Local CM Q_KEY	See section 12.7.8	8	32
Remote QPN/EECN	See section 12.7.37	12	24
Remote CM Response Timeout	See section 12.7.4	15	5
(reserved)		15[5]	3
(reserved)		16	32
Alternate Local Port LID	See section 12.7.11	20	16

Table 95 LAP Message Contents

Field	Description	Byte [Bit] Offset	Length, bits
Alternate Remote Port LID	See section 12.7.23 .	22	16
Alternate Local Port GID	See section 12.7.10 .	24	128
Alternate Remote Port GID	See section 12.7.22 .	40	128
Alternate Flow Label	See section 12.7.18	56	20
(reserved)		58[4]	4
Alternate Traffic Class	See section 12.7.17	59	8
Alternate Hop Limit	See section 12.7.19	60	8
Alternate Packet Rate (IPD)	See section 12.7.25	61	8
Alternate SL	See section 12.7.16	62	4
Alternate Subnet Local	See section 12.7.7	62[4]	1
(reserved)		62[5]	3
Alternate Local ACK Timeout	See section 12.7.34	63	5
(reserved)		63[5]	3
Private Data	See section 12.7.35	64	1344

12.8.2 APR - ALTERNATE PATH RESPONSE

APR is sent in response to a **LAP** request. **MRA** may be sent to allow processing of the **LAP**.

Table 96 APR Message Contents

Field	Description	Byte[Bit] Offset	Length, bits
Local Communication ID	See section 12.7.1 .	0	32
Remote Communication ID	See section 12.7.2 .	4	32
Additional Information		8	576
AP status	See section 12.8.2.1	80	4
(reserved)		80[4]	4
Private Data	See section 12.7.35	81	1208

12.8.2.1 AP STATUS

Value	Meaning
0	Alternate path information loaded
1	Invalid Communication Instance tuple
2	Alternate paths not supported. Alternate path parameters were not checked.
3	Alternate paths are supported and all Alternate Path parameters are valid, but the failover port was rejected for some other reason.
4	Alternate path information rejected - redirect
5	Proposed alternate path matches current primary path

If **AP Status** is "Alternate path information rejected - redirect", the **Additional Information** field contains a ClassPortInfo data structure as described in section 13.4.8.1 ClassPortInfo on page 619. The **LAP** sender may send a new **LAP** proposing the alternate path indicated by the ClassPortInfo.

12.9 STATE TRANSITION DIAGRAMS FOR COMMUNICATION ESTABLISHMENT AND RELEASE

The diagrams in this section detail all valid states and state transitions in the IBA communication establishment and release protocols. Section 12.10 contains ladder diagrams which illustrate various paths through this state diagram.

The InfiniBand™ communication establishment and communication release protocols are structured so that they will always run to completion in a bounded amount of time. “Completion” for the communication establishment protocol means that the communication will either be established, or else the state of all parties involved in the communication will revert to idle as if no communication had ever been established. “Completion” for the communication release protocol means that the communication is released; this protocol never fails to run to completion.

12.9.1 DIAGRAM DESCRIPTION

There is only one communication establishment protocol for InfiniBand™, with different messages used for different scenarios. The state diagrams are broken into an “active side” and a “passive side”. The active side of the protocol is the side that is trying to initiate a transition out of one of the terminal states (Idle and Established). The passive side of the protocol is the side that is responding to the active side.

12.9.2 INVALID STATE INPUT HANDLING

In many of the states of the InfiniBandTM communication establishment and release protocols, there is a defined set of input messages that can legally be received while in that state. The general rule for handling input messages that cannot be legally received and acted upon while in that state is to ignore them. A CM shall not retry the **REQ**, **REP**, or **DREQ** messages more than the number of times specified by **REQ:Max CM Retries**.

12.9.3 TIMEOUTS

A lost or dropped message will ultimately result in a timeout. Since all parties will ultimately return to the idle state, there is no correctness requirement to do retries of a message send as a result of a timeout, although it is recommended. Senders of retried messages may not modify the contents of the messages between retries.

In the following state diagrams, "Timeout" represents a Response Timeout. Service Timeouts are specifically noted.

12.9.4 STATE DIAGRAM NOTES

All **REJ**s, sent or received, cause a return to IDLE(active) or LISTEN(passive), possibly through the TimeWait state (see section 12.9.8.4).

In the Active Communication Establishment diagram, the transition from the Peer Compare state to the Passive REQ_Rcvd state only happens if the ServiceID in the **REQ** received is the same as the ServiceID in the **REQ** that was sent. (See section 12.10.4 for details). Otherwise, a new connection establishment instance shall be started.

The ServiceID implicitly defines whether the service is client/server or peer to peer, but the server application must inform its CM so that the CM will handle the inbound **REQ** correctly.

12.9.5 COMMUNICATION ESTABLISHMENT AND RELEASE - ACTIVE

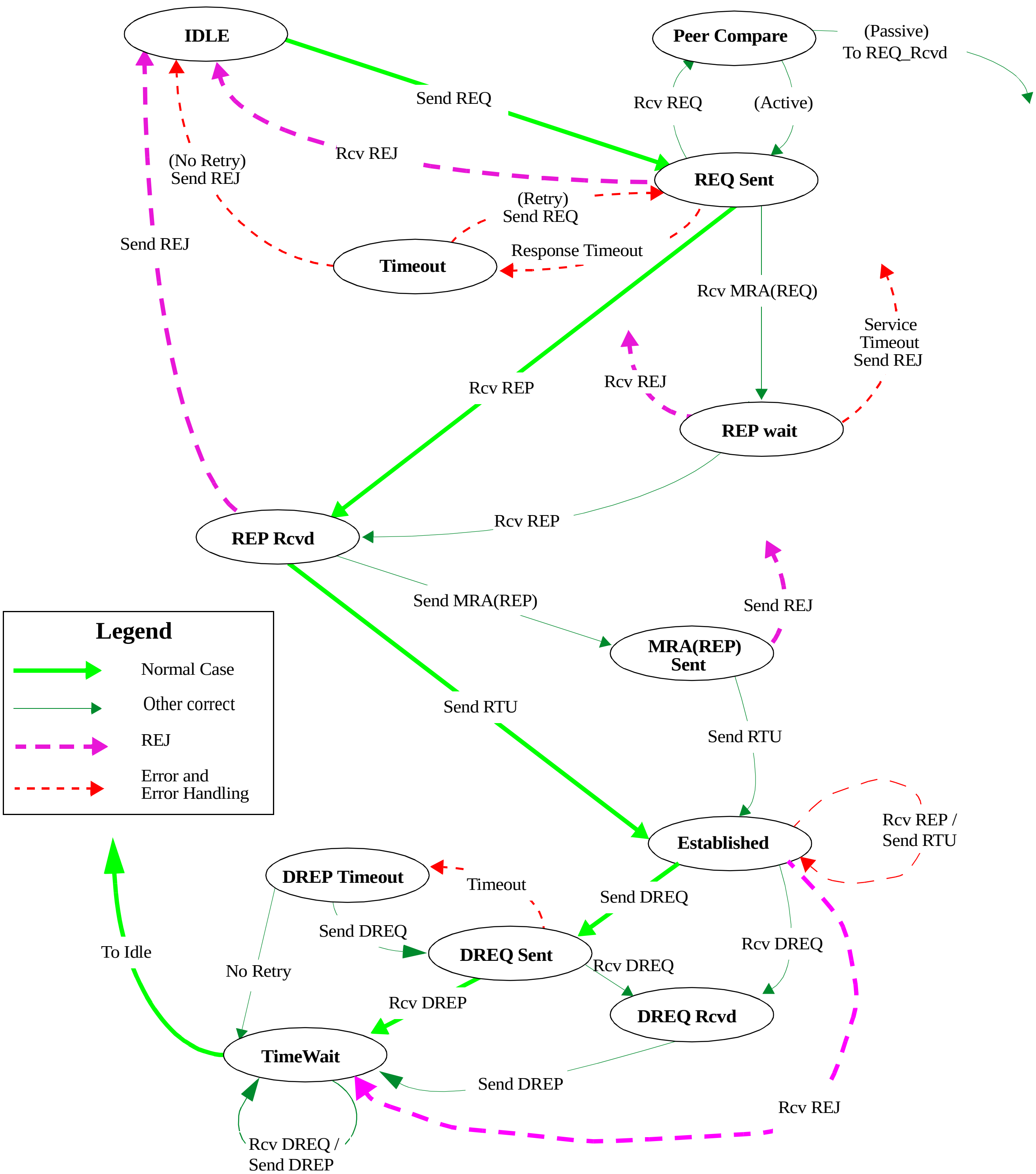


Figure 129 Communication Establishment (Active Side)

12.9.6 COMMUNICATION ESTABLISHMENT - PASSIVE

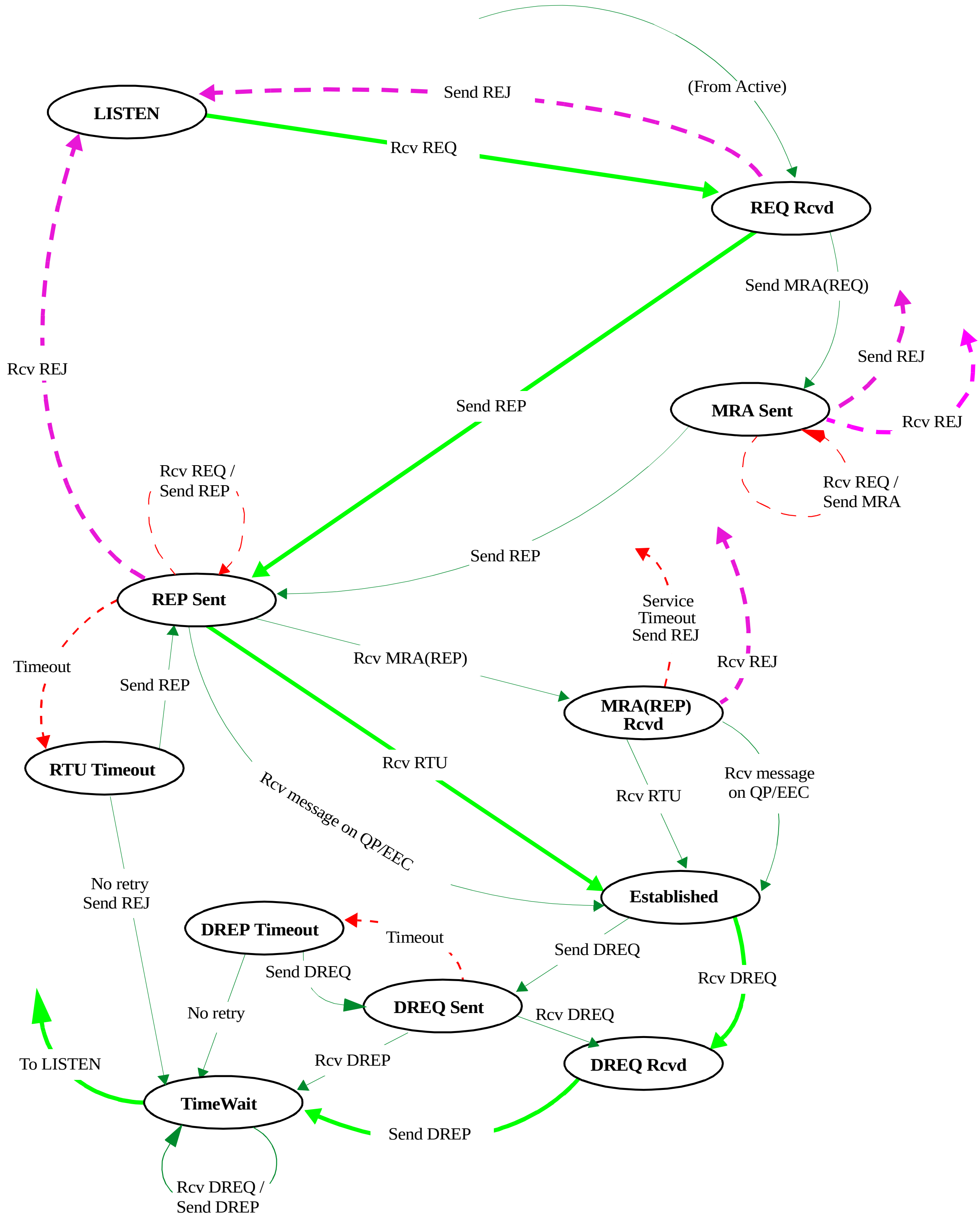


Figure 130 Communication Establishment(Passive Side)

12.9.7 State and Transition Definitions

The following tables define each state and the possible transitions from the state.

These tables define the protocol, and take precedence in the case of a conflict with the state diagrams.

In this table, "CEP" (Channel EndPoint) means QP or EEC, as appropriate.

12.9.7.1 Active States

CM State	Event	Action/Transition Sequence
IDLE		
	Entry	*CEP to Reset*
	Send REQ	Send REQ / CM to REQ Sent / *CEP to Initialized*
	(default)	None
REQ Sent		
	Receive REP	CM to REP Rcvd / *CEP to Ready to Receive*
	Receive REQ	IF (ServiceIDs match) to Peer Compare
	Receive MRA(REQ)	CM to REP wait
	Response Timeout	CM to Timeout
	Receive REJ	CM To IDLE / *CEP to Error*
	(default)	None
Peer Compare		

	Entry	IF (local CA GUID higher than remote CA GUID)
		CM to REQ Sent
		ELSE
		CM to Passive:REQ Rcvd

REP wait		
	Receive REP	CM to REP Rcvd / *CEP to Ready to Receive*
	Service Timeout	Send REJ / CM to IDLE / *CEP to Error*
	Receive REJ	CM to IDLE / *CEP to Error*

CM State	Event	Action/Transition Sequence
	(default)	None
REP Rcvd		
	Send RTU	Send RTU / CM to Established / *CEP to Ready to Send*
	Send MRA(REP)	CM to MRA(REP) Sent
	Send REJ	CM to IDLE / *CEP to Error*
	(default)	None
MRA(Rep) sent		
	Send RTU	Send RTU / CM to Established / *CEP to Ready to Send*
	Send REJ	CM to IDLE / *CEP to Error*
	Receive REJ	CM to IDLE / *CEP to Error*
	(default)	None
Established		
	Receive DREQ	CM to DREQ Rcvd / *CEP to Error*
	Send DREQ	CM to DREQ Sent / *CEP to Error*
	Receive REQ	See section 12.9.8.3.1
	Receive REP	Send RTU
	Receive REJ	CM to TimeWait
	(default)	None
DREQ Sent		
	Timeout	CM to DREP Timeout
	Receive DREQ	CM to DREQ Rcvd
	Receive DREP	CM to TimeWait
	(default)	None
DREQ Rcvd		
	Send DREP	CM to TimeWait
	(default)	None
TimeWait		
	Entry	CM: Start Timer / *CEP to Reset*

CM State	Event	Action/Transition Sequence
	Receive DREQ	CM: Send DREP (if Max CM Retries not exceeded)
	Timer Expiration	CM to IDLE
	(default)	None
Timeout		
	Entry (Retry) (Max Retries not exceeded)	Send REQ / CM to REQ Sent
	Entry (No Retry)	Send REJ / CM to IDLE / *CEP to Error*
DREP Timeout		
	Entry (Retry) (Max Retries not exceeded)	Send DREQ / CM to DREQ Sent
	Entry (No Retry)	CM to TimeWait
	(default)	None

12.9.7.2 Passive States

State	Event	Action/Transition Sequence
LISTEN		
	Entry	*CEP to Reset*
	Receive REQ	CM to REQ Rcvd / *CEP to Initialized*
	(default)	None
REQ Rcvd		
	Send REP	Send REP / CM to REP Sent / *CEP to Ready to Receive*
	Send MRA(REQ)	CM to MRA Sent
	Send REJ	CM to LISTEN / *CEP to Error*
	(default)	None
MRA sent		
	Send REP	Send REP / CM to REP Sent / *CEP to Ready to Receive*
	Send REJ	CM to LISTEN / *CEP to Error*
	Receive REJ	CM to LISTEN / *CEP to Error*

State	Event	Action/Transition Sequence
	Receive REQ	Send MRA
	(default)	None
REP Sent		
	Receive RTU	CM to Established / *CEP to Ready to Send*
	Receive MRA(REP)	CM to MRA(REP) Rcvd
	Receive message on service CEP	CM To Established / *CEP to Ready to Send*
	Receive REJ	CM to Listen / *CEP to Error*
	Timeout	CM to RTU Timeout
	Receive REQ	Send REP
	(default)	None
MRA(Rep) rcvd		
	Receive RTU	CM to Established / *CEP to Ready to Send*
	Service Timeout	Send REJ / CM to TimeWait / *CEP to Error*
	Receive message on service CEP	CM To Established / *CEP to Ready to Send*
	Receive REJ	CM to LISTEN / *CEP to Error*
	(default)	None
Established		
	Send DREQ	CM to DREQ Sent / *CEP to Error*
	Receive DREQ	CM to DREQ Rcvd / *CEP to Error*
	Receive REQ	See section 12.9.8.3.1
	(default)	None
DREQ Rcvd		
	Send DREP	CM to TimeWait
	(default)	None
DREQ Sent		
	Timeout	CM to DREP Timeout
	Receive DREQ	Send DREP / CM to TimeWait
	Receive DREP	CM to TimeWait

State	Event	Action/Transition Sequence
	(default)	None
RTU Timeout		
	Retry REP	CM to REP Sent
	No Retry	Send REJ / CM to TimeWait / *CEP to Error*
	(default)	None
TimeWait		
	Entry	CM: Start Timer / *CEP to Reset*
	Receive DREQ	CM: Send DREP (if Max CM Retries not exceeded)
	Timer Expiration	CM to IDLE
	(default)	None
DREP Timeout		
	Entry (Retry) (Max Retries not exceeded)	Send DREQ / to DREQ Sent
	Entry (No Retry)	CM to TimeWait
	(default)	None

12.9.8 STATE DETAILS

12.9.8.1 TIMEOUT

A message may be re-sent no more than **REQ**:Max CM Retries , but there is no requirement that it be re-sent that many times.

12.9.8.2 RTU TIMEOUT

If the Passive agent sends **REP** but does not receive either an **RTU** or a message on the CEP (QP or EEC, as appropriate), it transitions to **RTU** Timeout. If it has not exceeded **REQ**:Max CM Retries , the Passive agent may resend **REP**.

12.9.8.3 ESTABLISHED

12.9.8.3.1 REQ RECEIVED

(RC, UC) A CM may receive a **REQ** specifying a remote QPN in "**REQ**:local QPN" that the CM already considers connected to a local QP. A local CM may receive such a **REQ** if its local QP has a stale connection, as described in section 12.4.1. When a CM receives such a **REQ** it shall abort the connection establishment by issuing **REJ** to the **REQ**. It shall then issue **DREQ**, with "**DREQ**:remote QPN" set to the remote QPN from the **REQ**, until **DREP** is received or Max Retries is exceeded, and place the local QP in the TimeWait state.

(RD) A CM may receive a **REQ** specifying a remote EECN in "**REQ**:local EECN" that the CM already considers connected to a local EEC. A local CM may receive such a **REQ** if its local EEC has a stale connection, as described in section 12.4.1. When a CM receives such a **REQ** it shall abort the connection establishment by issuing **REJ** to the **REQ**. It shall then issue **DREQ**, with "**DREQ**:remote EECN" set to the remote EECN from the **REQ**, until **DREP** is received or Max Retries is exceeded, and place the local EEC in the TimeWait state.

12.9.8.4 TIMEWAIT

The **PathRecord:PacketLifeTime** (section 15.2.5.17 PathRecord) field defines the maximum time that a packet can exist in the fabric.

The TimeWait timer shall be set to twice the **PathRecord:PacketLife-Time** value plus the remote's Ack Delay.

The CM is responsible for placing QPs/EECs in the TimeWait state, for maintaining them in that state for a period not less than the TimeWait period, and for removing them afterward.

Receipt of a **DREQ** while in the TimeWait state shall not affect the Time-Wait timer.

12.9.8.5 MESSAGE RECEIPT ACKNOWLEDGMENT (MRA)

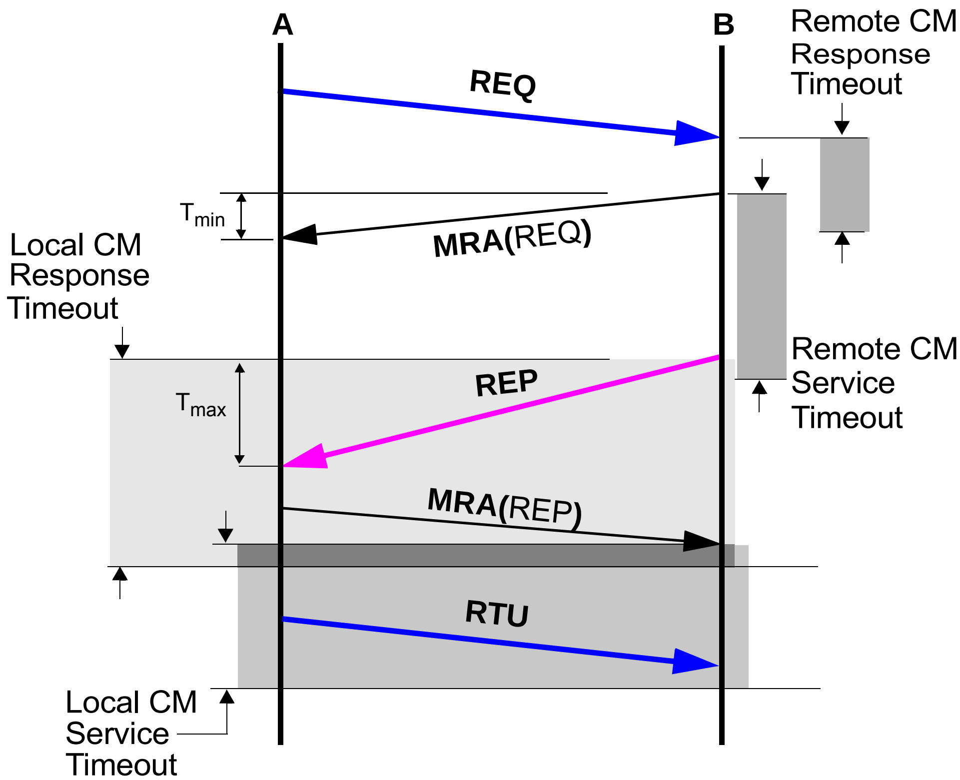


Figure 131 MRA Example

Figure 131 illustrates the use of the **MRA** message in a CM message exchange. 'Local' and 'Remote' are with respect to 'A'. Because B cannot return a **REP** or **REJ** to A within the **Remote CM Response Timeout**, B sends an **MRA(REQ)**, notifying A that B has received the REQ message and is processing it. The **MRA(REQ)** contains B's **CM Service Timeout** value. B completes its processing and sends the **REP** message to A before the expiration of the **Remote CM Service Timeout**.

Because packet flight times may differ due to fabric congestion, (e.g., the **MRA** may travel in the minimum possible time, and the **REP** in the maximum time, as shown by T_{min} and T_{max}), A shall allow an additional PacketLifeTime for the **REP** to arrive.

When A receives the **REP**, it realizes that the required processing will not allow it to transmit a **REJ** or **RTU** soon enough to arrive at B before the **Local CM Response Timeout** expires, so it sends an **MRA(REP)** containing its CM Service Timeout value. When it completes the REP processing, A sends the **RTU**, which arrives before the **Local CM Service Timeout** expires.

Once an **MRA** is received, the CM shall not re-send the message acknowledged by the **MRA** sooner than the period of time represented by the applicable CM Service Timeout period plus PacketLifeTime.

12.9.8.6 TIMEOUTS AND RETRIES

In the communication establishment protocol, the sending of the **REQ**, **REP** and **DREQ** messages may be retried by the sender. The retry happens after the sender fails to receive a response message from the recipient within the appropriate response timeout period.

For the **REQ** message, the Remote CM Response Timeout period is the recipient's "turn-around" time. The **REQ** sender may consider the **REQ** (or response) lost after (2*PacketLifeTime + Remote CM Response Timeout). Upon receiving a **REQ,** the recipient must send a **REP**, **REJ**, or **MRA** by the Remote CM Response Timeout. The Service Timeout period begins when the **MRA** is sent, and a **REJ** or **REP** must be sent before it expires.

The Local CM Response Timeout tells the **REP** sender how long to wait for an **MRA**, **REJ**, or **RTU**. The Local CM Response Timeout value includes the round trip flight time. If the **REP** sender receives an **MRA**, it can expect the **REJ** or **RTU** within (Local CM Service Timeout + PacketLifeTime) after the **MRA**'s arrival.

The response timeout period for the **DREQ** message is the Local CM Response Timeout present in the original **REQ** message.

When the sender retries a message send, the recipient can potentially receive multiple copies of the same message. The recipient of a **REQ** (or **REP**) message should determine the amount of time it has to send a response based upon when it received the latest **REQ** (or **REP**) message; the remaining time it has to reply is thus reset back to the full response timeout period each time it receives a new **REQ** (or **REP**) for the same connection establishment attempt.

If the sender of a **REP** message receives another **REQ** message for the same connection establishment attempt, after it resends the **REP** message it should reset its response timeout period back to the full Local Response Timeout period that it received in the **REQ** message.

12.9.9 CONNECTION STATE

Communication Managers shall maintain the following information for the life of a connection:

- Local Communication ID
- Remote Communication ID

- Local CM Response Timeout
- Remote CM Response Timeout
- Local QPN / Local EECN
- Remote QPN / Remote EECN
- Remote LID / Remote GID
- Max CM Retries

12.10 COMMUNICATION ESTABLISHMENT LADDER DIAGRAMS

The following ladder diagrams show the message exchanges for various communication establishment scenarios. These are not applicable to Unreliable Datagrams (see section 12.11 for Service ID Resolution).

12.10.1 ACTIVE CLIENT TO PASSIVE SERVER - BOTH CLIENT AND SERVER ACCEPT COMMUNICATION

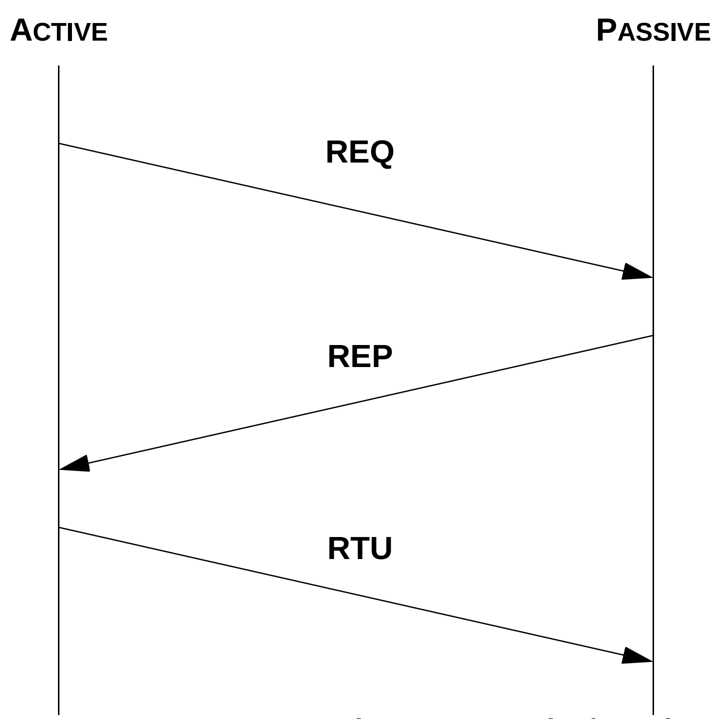


Figure 132 Active/Passive, Both Accept

For RC and UC service, the above exchange establishes the connection.

For RD service, the above exchange must be performed

- To establish a Reliable Datagram Channel between two EECs
- To resolve a Service ID and associate QPs for possible use over an existing RDC

How cooperating applications exchange information on additional available QPs is specific to the applications.

12.10.2 Active Client to Passive Server - Server Rejects Communication

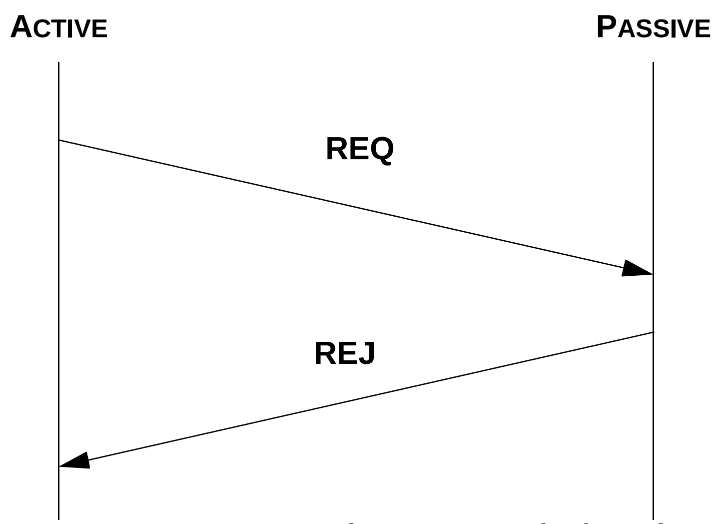


Figure 133 Active/Passive, Server Reject

The above exchange occurs when the passive server cannot or will not perform the requested action. The **REJ** message contains the reason why the action was not performed.

12.10.3 ACTIVE CLIENT TO PASSIVE SERVER - CLIENT REJECTS COMMUNICATION

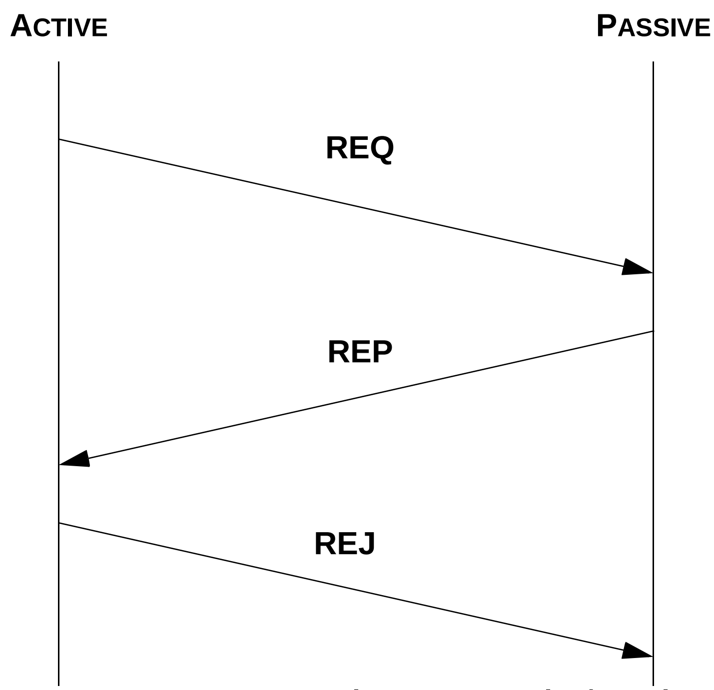


Figure 134 Active/Passive, Client Reject

The above exchange occurs when the requesting client decides not to continue with the requested action. (An example is a client that requires Automatic Path Migration support not provided by the server.) The **REJ** message contains the reason why the action was not continued.

12.10.4 PEER TO PEER - BOTH ACCEPT COMMUNICATION

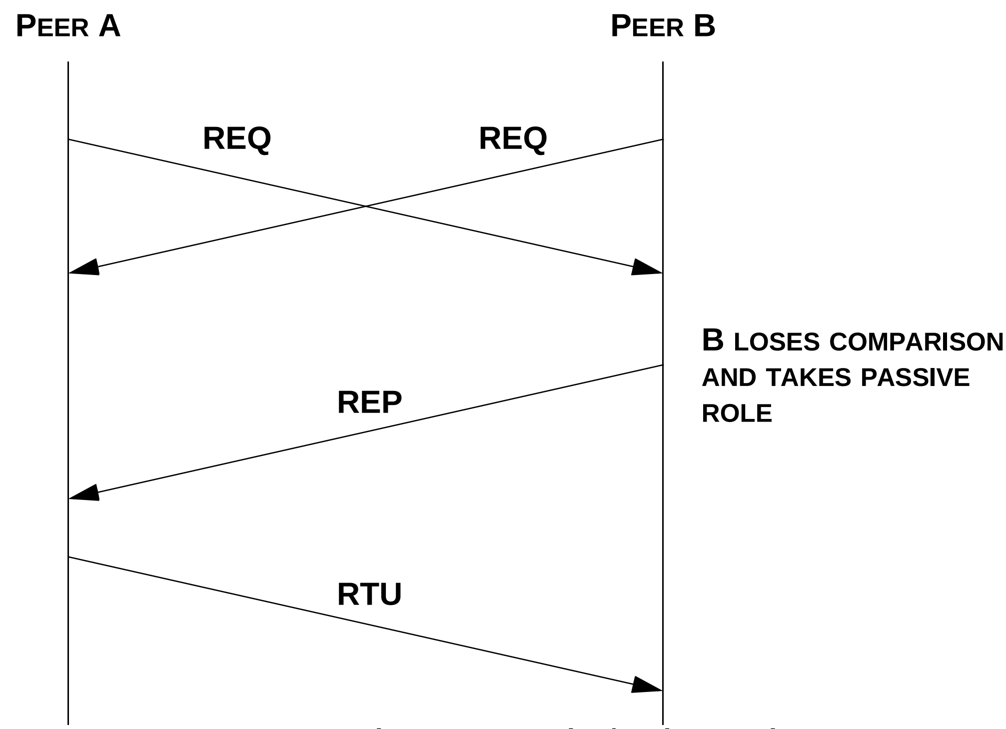


Figure 135 Active/Active, Both Accept

The above exchange occurs when two peer entities attempt communication. In this case, the ServiceID in both **REQ** messages is the same. The ServiceID implicitly defines whether the service is client/server or peer to peer, but the application must inform the CM so that the CM will handle the inbound **REQ** correctly. (For instance, if a client on A wishes to establish a connection to a server on B at the same time a client on B wishes to establish a connection to the same service on A, each CM must know that, because the services are client/server, there are two active/passive connection instances in progress, and not a single active/active instance.)

Peer A and Peer B compare their CA (Channel Adapter) GUIDs, treating each as a big-endian value, to decide which party will take the active side of the CM protocol. The peer with the numerically smaller GUID assumes the passive role in the remainder of the communication establishment protocol.

If the CA GUIDs match (e.g., two processes using the same CA), the **REQ:Local QPN** fields shall be compared, treating each as a big-endian value, with the smaller QPN taking the passive role.

CM shall not be used to establish "loopback" channels on a single QP.

12.10.5 ACTIVE PEER TO ACTIVE PEER - PASSIVE REJECTS COMMUNICATION

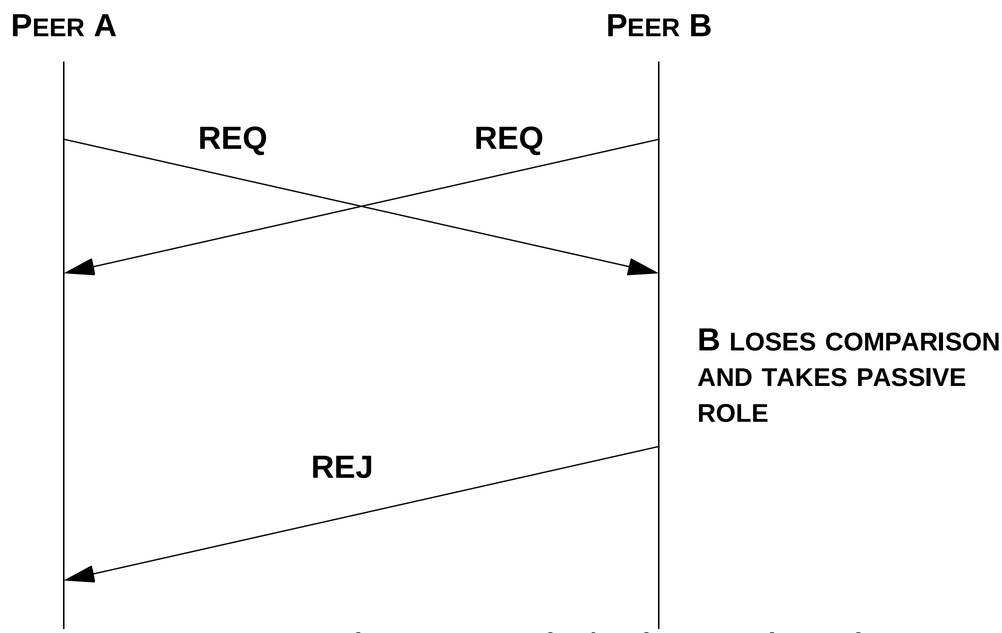


Figure 136 Active/Active, Passive Reject

The above exchange occurs when the ’losing’ peer decides not to continue the requested action. The **REJ** message contains the reason the action was not continued.

12.10.6 ACTIVE PEER TO ACTIVE PEER - ACTIVE REJECTS COMMUNICATION

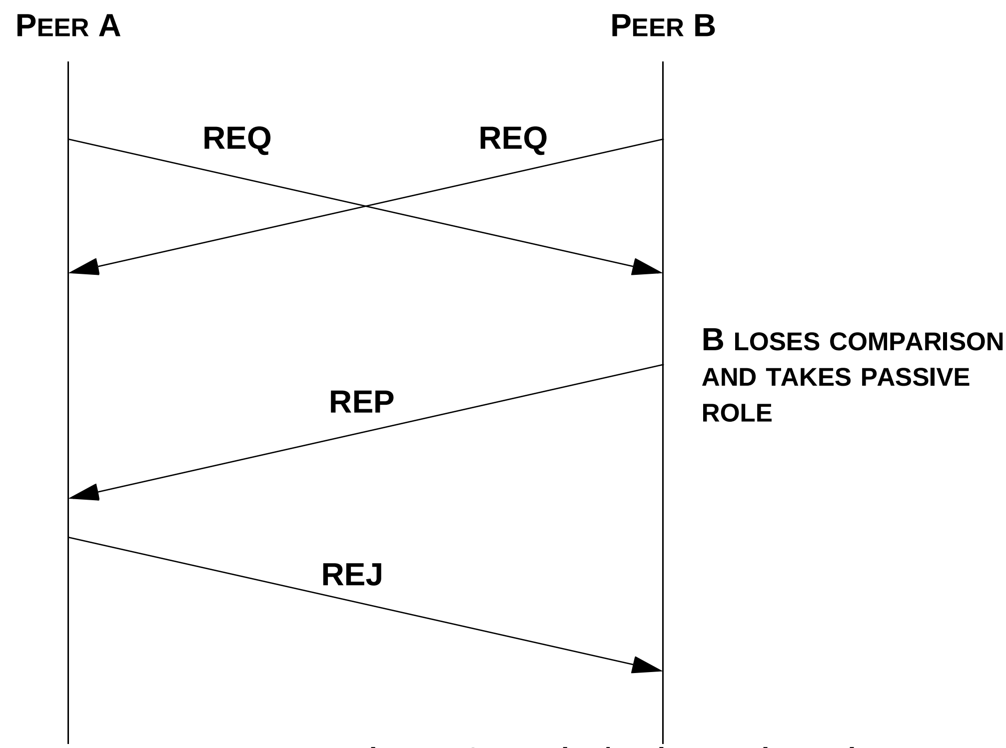


Figure 137 Active/Active, Active Reject

The above exchange occurs when the when the 'winning' peer decides not to continue with the requested action. The **REJ** message contains the reason why the action was not continued. The peer receiving the **REJ** message returns to the IDLE state.

12.10.7 ACTIVE CLIENT TO PASSIVE SERVER WITH REDIRECTOR - ALL ACCEPT COMMUNICATION

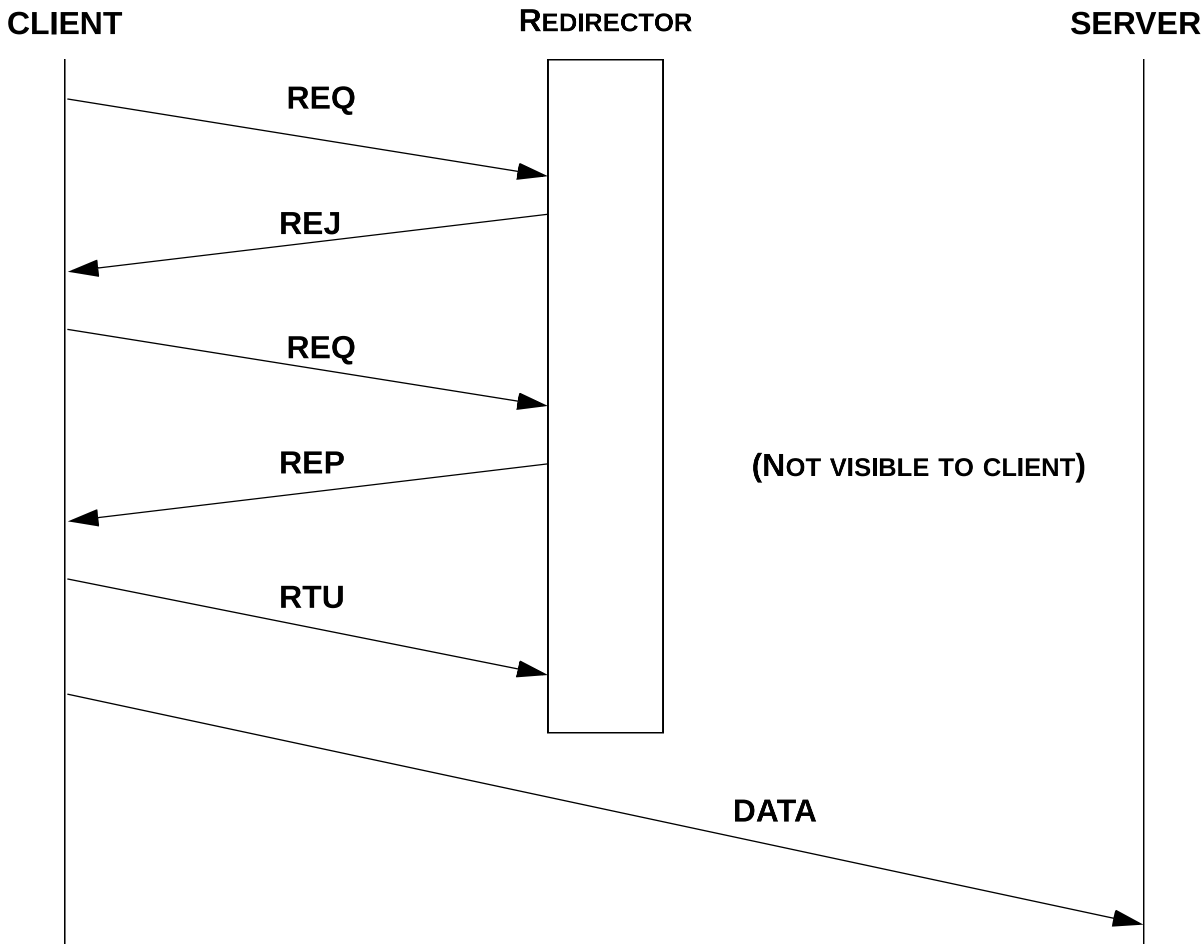


Figure 138 Redirection, Accepted

A redirector is a CM that provides CM services on behalf of an entity supported on a port other than the redirector CM's port. The port information for both endpoints is explicit in the **REQ** and **REP** messages, allowing the redirector to manage connections as a proxy for another entity.

The above exchange occurs when the redirector sends **REJ** with the Status "Port Redirection", indicating that the requested ServiceID is available at a different port. The requesting client (using a new Local Communication ID) sends a new **REQ** proposing the port specified in the **REJ**. All CM messages are exchanged between the client CM and the redirector, but traffic over the established connection goes between the client and the server.

12.10.8 COMMUNICATION RELEASE

The following ladder diagram shows the message exchange for communication release.

Communication release as illustrated in this section is ungraceful. Upon receipt of a Disconnect Request, each CM shall cause the affected QP to be placed into the error state, causing pending work requests to complete with the Flush error status.

Consumers are free to define and execute a more graceful communication release protocol that allows for an orderly shutdown of communications. Any such protocol shall utilize the communication release protocol illustrated below after the termination of normal message processing.

12.10.8.1 DISCONNECT REQUEST

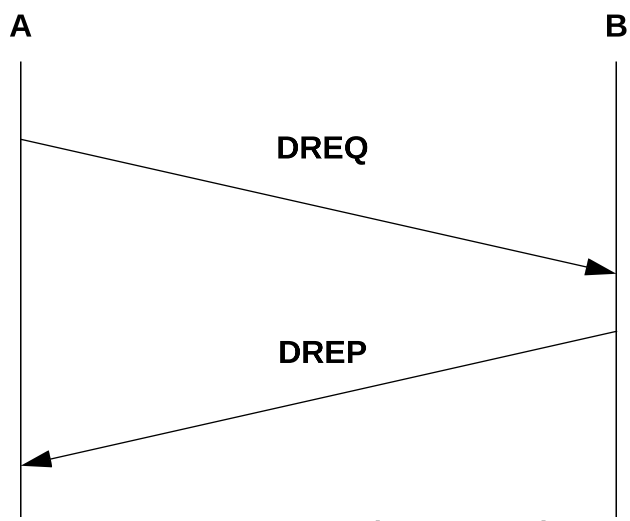


Figure 139 Disconnect Request

Because the **DREQ** and **DREP** travel out of band relative to normal communications traffic, how operations currently in progress will be completed cannot be predicted.

12.11 SERVICE ID RESOLUTION PROTOCOL

Service ID Resolution (SIDR) provides a way for users of Unreliable Datagram (UD) service to determine a Queue Pair on the target port that supports a given Service ID.

GSAs shall support this protocol if non-management services are provided on the Channel Adapter at other than fixed QPNs. If this protocol is not supported, the status "The method/attribute combination is not supported" shall be returned, as described in section Table 102 MAD Common Status Field Bit Values .

The protocol consists of a single request and a single reply, using unreliable Management Datagrams (MADs) targeted to the GSI. SIDR messages are of the Communication Management class.

If the SIDR response returns a valid QPN, the returned QPN shall be in the partition identified by the P_Key in the header of the SIDR Request.

12.11.1 SIDR_REQ - SERVICE ID RESOLUTION REQUEST

SIDR_REQ requests that the recipient return the information necessary to communicate via UD messages with the entity specified by **SIDR_REQ:ServiceID**.

Table 97 SIDR_REQ Message Contents

Field	Description	Byte[Bit] Offset	Size, bits (Values)
RequestID	See section 12.11.1.1	0	32
(reserved)		4	32
ServiceID	See section 12.11.1.2	8	64
Private Data	See section 12.11.1.3	16	1728

12.11.1.1 REQUESTID

A 32-bit value identifying the request. The target of the request shall return this value unchanged in the **SIDR_REP** message. If a **SIDR_REQ** message is re-sent, the sender shall send the same RequestID.

12.11.1.2 SERVICE ID

The Service ID that the sender wishes to have resolved. See section 12.7.3 for more information.

12.11.1.3 PRIVATE DATA

Data that is opaque to the SIDR protocol for use by the requester and responder. For example, some systems may require that the PrivateData area contain an authorization key before reporting the QP supporting certain ServiceIDs.

12.11.2 SIDR_REP - SERVICE ID RESOLUTION RESPONSE

SIDR_REP returns the information necessary to communicate via UD messages with the entity specified by **SIDR_REQ:ServiceID**.

Table 98 SIDR_REP Message Contents

Field	Description	Byte [Bit] Offset	Length, bits
RequestID	See section 12.11.1.1	0	32
QPN	See section 12.11.2.2	4	24
Status	See section 12.11.2.1	7	8
ServiceID	See section 12.11.1.2 .	8	64
Q_Key	See section 12.11.2.3	16	32
ClassPortinfo	This is a ClassPortinfo data structure as documented in section 13.4.8.1, describing where to redirect the **SIDR_REQ**. This field is only valid if the Status field indicates the request should be redirected.	20	576
Private Data	See section 12.11.1.3	92	1120

12.11.2.1 STATUS

The Status field tells whether the QPN field is valid, and if not valid, the reason a valid QPN was not provided.

0	QPN and Q_Key are valid
1	Service ID not supported
2	Rejected by Service Provider
3	No QP available
4	Request should be redirected to the endpoint specified by ClassPortinfo. QPN and Q_Key are not valid.
5-255	Reserved

12.11.2.2 QPN

The QPN of the local QP on which the requested Service ID is supported. (Only valid if so indicated by Status field).

12.11.2.3 Q_KEY

The Q_Key for the QP returned in **QPN**.

12.11.3 PATH INFORMATION

The information returned in the **SIDR_REP** message is insufficient, by itself, to create a usable address handle. Specifically, the values for **PathRecord:Mtu** and **PathRecord:Rate** are required except when sending packet payloads no larger than the minimum PMTU, or when transmitting on a minimum-width link, respectively. These values are available through Subnet Administration (see section 15.2.5.17).

CHAPTER 13: MANAGEMENT MODEL

13.1 INTRODUCTION

IBA management is built on top of four fundamental concepts. These include:

- Management entities
- Agents.
- A messaging scheme.
- A collection of specific messages including message content, and related behaviors.

An agent is a conceptualization of a body of low level functionality embedded in all channel adapters, switches, and routers, which provides the means to set and query various parameters internal to the channel adapter, switch, or router.

Managers and interested parties are conceptualizations of high level bodies of functionality which provide for controlling or examining various aspects of subnet or fabric configuration and operation.

The messaging scheme provides for intercommunication between managers or interested parties and agents, and, in some cases, between agents. The messaging scheme specifies the basic message types and interfaces through which agents and managers exchange information.

Finally, specific messages and message sequences are defined in terms of message content and associated required behaviors. Messages are grouped into classes according to the type of management activity the messages support.

The specification of management operations is done from the viewpoint of specifying messages that may appear on the wire and specifying behaviors associated with those messages. The appearance of a message at a port implies a required action and, possibly, response. Additionally, the appearance of a message on the wire implies behavior of the entity that caused the message to be emitted. In particular, the behavior requirements in certain areas (e.g. subnet management, see 14.4 Subnet Manager on page 687) imply the existence of certain entities (e.g. a subnet manager) which embody required behaviors with respect to the origination and consumption of various messages.

Various conceptualizations are used in specifying behaviors. However, the use of such conceptualizations in this and other management related chapters is purely a descriptive artifice. The conceptualizations themselves, do not convey normative requirements. Normative requirement specification is done by, and only by, specification of message formats and associated required behaviors. Finally, while some conceptualizations may suggest certain implementations, implementations are outside of the scope of the specification and no specific implementation is implied.

13.2 Assumptions, and Scope

13.2.1 Assumptions

There are certain assumptions that underlie the management mechanisms specified herein. Proper operation of the management mechanisms and fulfillment of the objectives underlying these specifications is predicated upon the validity of these assumptions. While the assumptions themselves are not part of the specification, they are an essential element of the framework in which these specifications apply.

- The management operations specified herein provide for a level of interoperability such that an SM from any vendor can manage a heterogeneous collection of IBA-compliant channel adapters, switches, and routers from any set of vendors. However, compatibility and interoperability among SMs from different vendors is not supported. Migration from one vendor's SM to another's by way of system re initialization, i.e., through a planned outage, is supported. Such migration assumes appropriate steps of transferring data between vendors' SMs have been accomplished prior to the re initialization.
- The management operations specified herein provide the means to conduct a variety of activities. Some of the mechanisms specified are optional. And, except as specifically stated, the specification of a means does not imply or require that the means be used. It is assumed that each fabric will be constructed, configured, and operated according to the needs of its user(s) and that constructors exercise diligence in selection of components to ensure the fabric possesses the characteristics required. For example, if a user requires multicast support but mixes components that do and do not support multicast, they may fail to achieve their requirements.

13.2.2 Scope

As noted above, a number of management classes are distinguished in the IBA management model. The classes include:

- Subnet management. Subnet management is the body of activity associated with discovering, initializing, and maintaining an IBA subnet. In addition, the subnet management sections specify methods for interfacing to a diagnostic framework for handling subnet and protocol errors. (See 14.2.5.14 VendorDiag on page

679). In the following sections a subnet manager will be denoted by SM while a subnet management agent will be denoted by SMA.

- Subnet administration (SA): Subnet administration provides a means for management entities and applications to obtain information about fabric configuration and operation. (See Chapter 15: Subnet Administration on page 701). In the following sections subnet administration will be denoted by SA.
- Communication management: Communication management provides the means to set up and manage communications between a pair of queue pairs or, in certain cases, to identify which queue pair to use for a certain service. (See Chapter 12: Communication Management on page 545 and 16.7 Communication Management on page 818). In the following sections a communications manager will be denoted by CM while a communications management agent will be denoted by CMA.
- Performance management: Performance management specifies a set of facilities for examining various performance characteristics of a fabric. (See 16.1 Performance Management on page 748). In the following sections a performance manager will be denoted by PM while a performance management agent will be denoted by PMA.
- Device management: Device management specifies the means for determining the kind and location of various kinds of devices on a fabric. (See 16.3 Device Management on page 793). In the following sections a device manager will be denoted by DM while a device management agent will be denoted by DMA.
- Baseboard management: Baseboard management specifies the means to effect, in-band (i.e. over the IBA fabric) low level system management operations. (See 16.2 Baseboard Management on page 781 and VOLUME 2, Chapter 13, Hardware Management). In the following sections a baseboard manager will be denoted by BA while a baseboard management agent will be denoted by BMA.
- SNMP tunneling: SNMP tunneling specifies mechanisms to support transport of SNMP operations through an IBA fabric. (See 16.4 SNMP Tunneling on page 804). In the following sections a SNMP tunneling agent will be denoted by SNMPA.
- Vendor specific: The vendor specific classes specify a basic framework within which a vendor can define vendor specific management communications and operations that are beyond the scope of the IBA. See 16.5 Vendor-specific on page 813 for architectural details relating to use of specific values.

- Application specific: The application specific classes specify a basic framework within which services can be defined which implement operations that are beyond the scope of the IBA. See 16.6 Application-specific on page 815 for architectural details relating to use of specific values.

As a notational convenience, the set of classes as listed above but excluding subnet management are referred to as General Services. When referring to general services managers, the notation GSM may be used. When referring to general services agents, the notation GSA may be used. According to the context in which it appears, GSM(s) or GSA(s) may refer to the group of all supported general services managers or agents on a channel adapter, switch, or router, or to any of the managers or agents in that group.

The IBA management services provided by the above classes support management of only the devices that comprise the IBA subnet. They do not support management of devices beyond the subnet. Specifically, they do not support management of tape drives, hard disk drives, network interfaces, etc. Mechanisms required to discover and power-manage devices that are accessed through an IBA channel adapter are provided within the above classes but provision of services specific to such devices is beyond the scope of the IBA.

IBA management provides a means of configuring and gathering information from IBA channel adapters, switches, and routers.The IBA Subnet Administration Service provides a means for other entities to determine the topology and configuration of the subnet. For example, operating systems or other higher level management entities may use IBA Subnet Administration services mechanisms to enforce operating system policies, or cluster policies, and so on, but such higher level entities and the policies they effect are outside the scope of IBA management services.

A variety of standards for communication of management information between managed elements and management applications exist today. These include SNMP, DMI, and CIM (Simple Network Management Protocol, Desktop Management Interface, and Common Information Model), as well as other standard and proprietary interfaces. Such standards may be layered on top of the IBA management model interfacing to it through services defined in the model. Alternatively, they may interface to IBA management elements through private interfaces. In either case, while the IBA management model provides means for such applications to obtain subnet topology and configuration information, such applications are outside the scope of IBA management.

Finally, the current IBA specification defines only the mechanisms required for proper operation of IBA fabrics and interoperation of IBA components. Specific applications, such as for enclosure management, can also be used in conjunction with non-IBA subsystems that are connected to the IBA subnet. Such applications may utilize the IBA subnet as a means of transport for the specific subsystem management data but such subsystem management services themselves are outside of the scope of the management services specified for IBA.

Figure 140 Management Model on page 599, below, depicts an example subnet indicating graphically the relationships among the IBA managed subnet and related services and higher level and lower level entities that may be found on an IBA fabric.

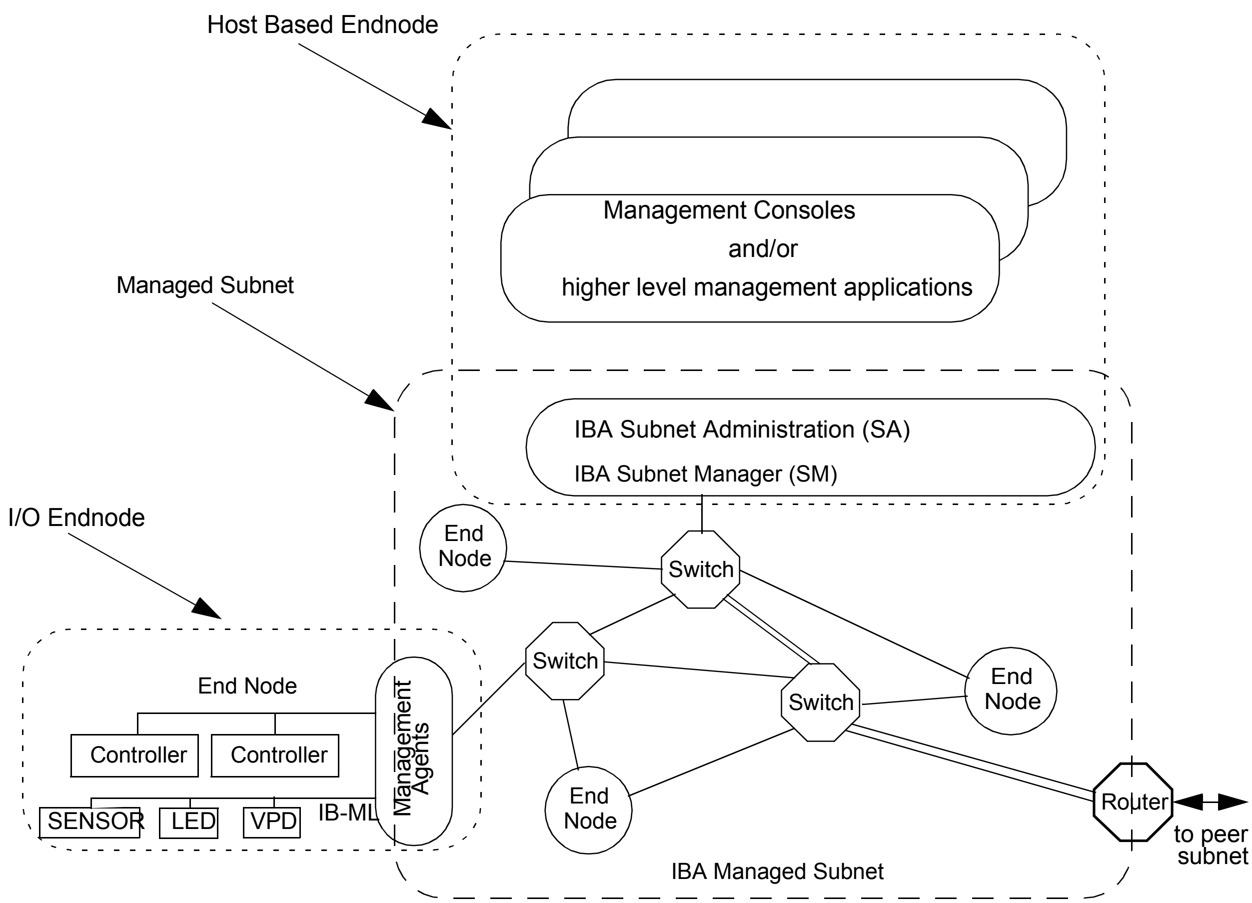


Figure 140 Management Model

This chapter provides an overview of the IBA management model, the management entities, and the corresponding interfaces. In addition this chapter defines requirements and specifies mechanisms common to all management activities. Subsequent chapters specify additional details associated with specific management classes. For each management class, the complete set of applicable requirements that must be satisfied and mechanisms that must be provided is the combination of those from this chapter with those in the corresponding class specific sections of the other chapters.

13.3 Managers, Agents, and Interfaces

13.3.1 Introduction

IBA Management is organized around abstract functional entities referred to as *managers* and *agents,* and, *interfaces*. Communication between managers and agents is performed through management messages referred to as Management Datagrams (MADs). MADs are exchanged using the unreliable datagram transport service as defined in 9.8.3 Unreliable Datagrams on page 355.

Managers are conceptual functional entities that effect control over fabric elements or provide for gathering information from fabric elements. In general, managers may reside anywhere in a subnet although class specific constraints on the manner in which they logically interface to the fabric medium (e.g. SMs use QP0, see 13.5.1 MAD Interfaces on page 630) may impose specific restrictions.

Agents are conceptual functional entities present in IBA channel adapters, switches, and routers that process management messages arriving at the ports of the IBA channel adapters, routers, and switches where they exist.The functionality represented by an agent effects required behaviors associated with MADs which arrive at the port or ports with which it is associated.

Abstractly, interfaces represent a target to which messages may be sent and through which messages will be processed or will be dispatched to an appropriate processing entity. For management interfaces, the associated processing entity is an agent or, in some cases, a manager. As such, an interface is a means to gain access to the functionality of agents and/or managers.

Management operations are divided into a set of classes. For a given class of activity, there is usually only a small number of managers on a subnet. Conceptually, for each supported class, there is one agent on each switch, channel adapter, and router on the IBA subnet.

Although the notions of agent, manager, and interface as described above, may suggest specific implementations, this specification only mandates behavior with respect to sourcing and sinking management messages, not how that behavior is achieved. The notion of an agent, manager, or interface, is a convenient descriptive artifice which encapsulates functional operations and behaviors associated with a particular class of activities. This specification does not require the existence of agents, managers, or interfaces per se. It does require that implementations exhibit the behaviors associated with the abstract agents, managers, and interfaces. How that is actually accomplished is implementation dependent.

The messages and behaviors relating to the subnet management class are further defined in 14.2 Subnet Management Class on page 642. This class uses specialized MADs referred to as Subnet Management Packets (SMPs).

Figure 141 on page 601 depicts a single subnet showing representative relationships among channel adapters, switches, subnet managers and agents.

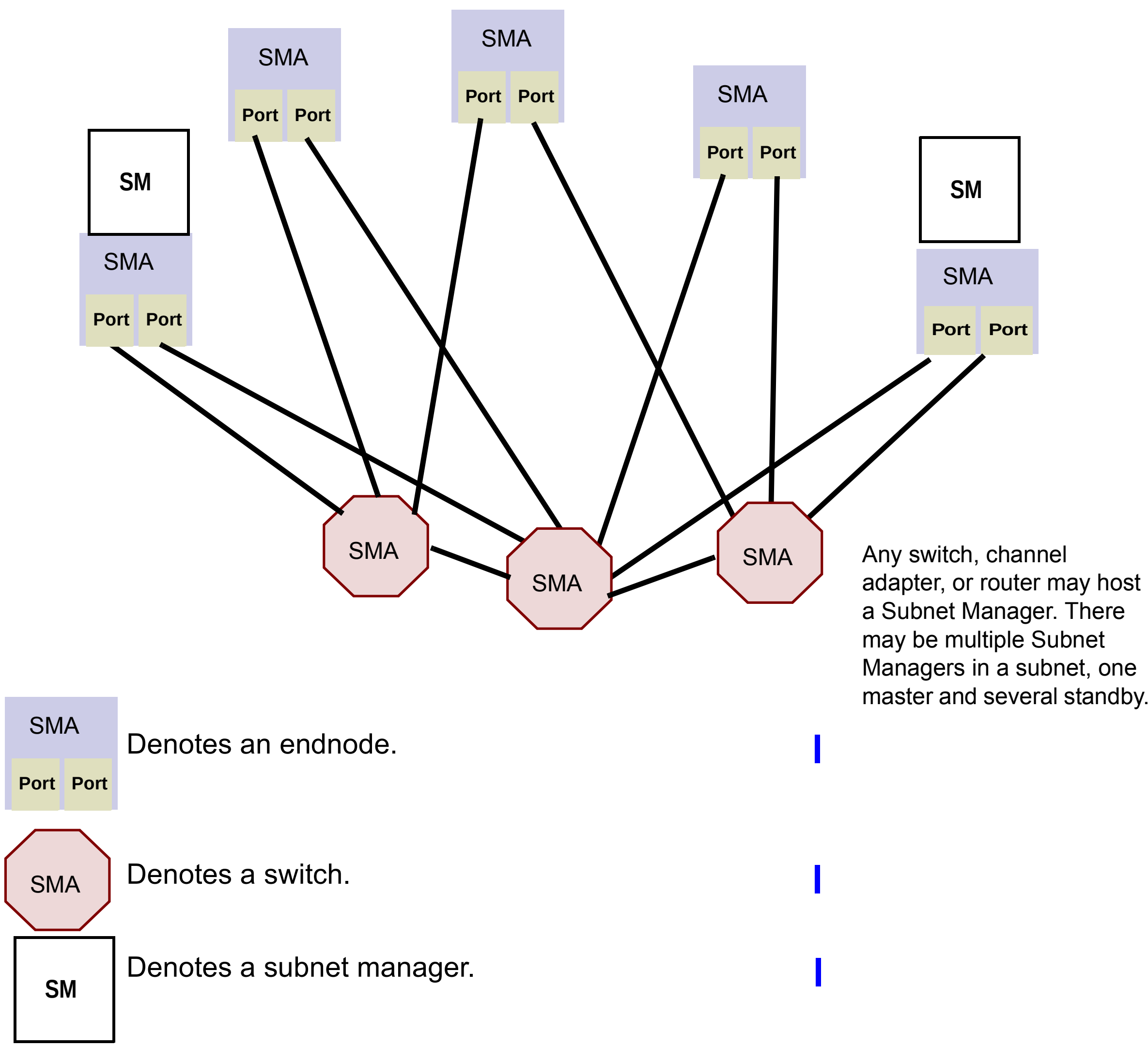


Figure 141 A Single Subnet Depicting Representative Subnet Manager/Agent Relationships

The messages and behaviors relating to the subnet administration class are further defined in 15.2 SA MADs on page 702 while the messages and behaviors relating to the other general services classes are further defined in the subsections of Chapter 16: General Services on page 748. These service classes use MADs referred to as General Services Management Packets (GMPs).

Figure 142 on page 602 depicts a single subnet showing representative relationships between general service class managers and corresponding agents.

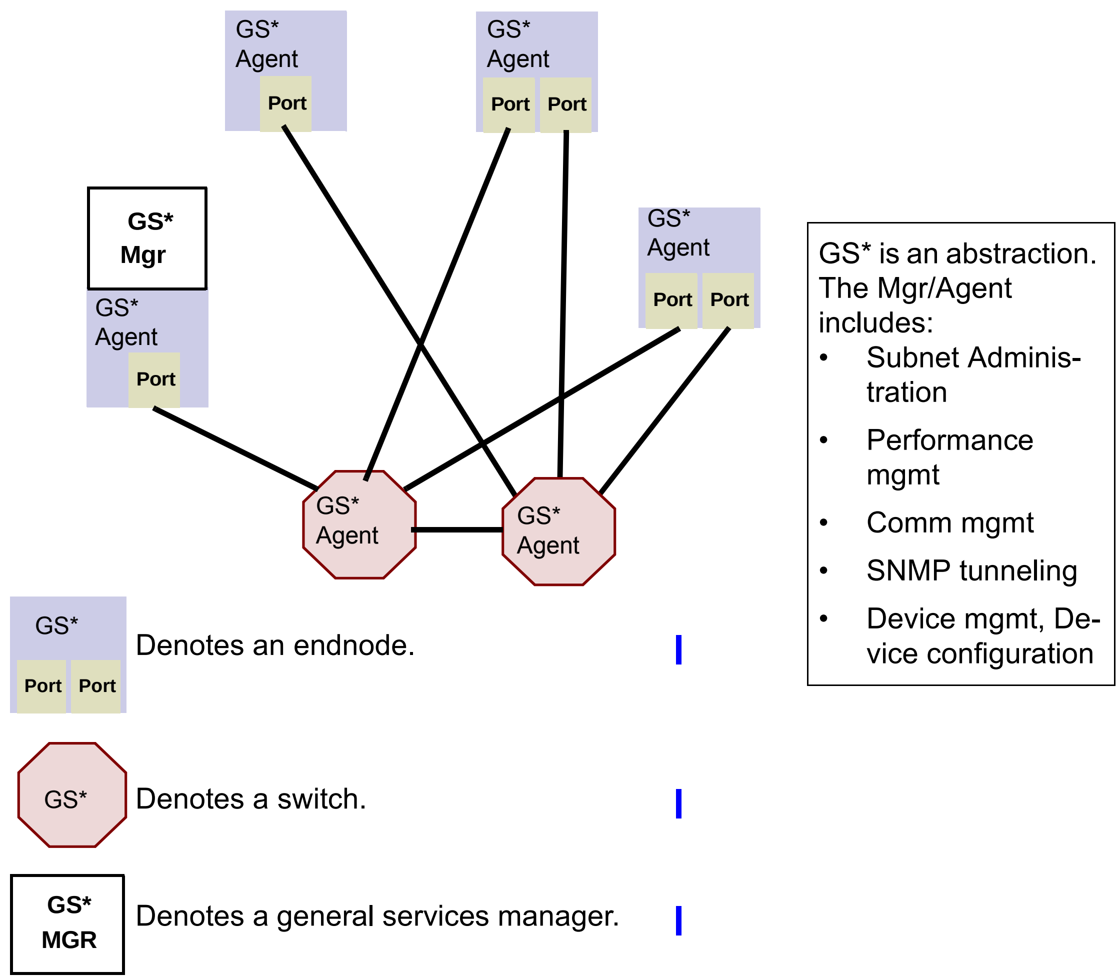


Figure 142 A Single Subnet Depicting Representative General Services Management/Agent Relationship

13.3.2 REQUIRED MANAGERS AND AGENTS

C13-1: Each subnet shall have at least one logical SM.

Logical SMs may be single physical entities or may consist of multiple, possibly distributed, cooperating physical entities which collectively effect the appearance of a single SM to CAs switches and routers on the subnet it manages.

If there is more than one entity capable of acting as a master SM, only one should function as a master SM during initialization.

See Chapter 14: Subnet Management on page 641 for additional specific requirements applicable to SMs during and after initialization.

There is a close relationship between SMs and SAs. This is described in 15.1.2 Relationship Between SA and the SM on page 702.

IBA version 1.0 does not otherwise mandate the existence of, the location of, or, operational characteristics of GSMs. The class specific sections of Chapter 16: General Services on page 748 define messages and agent behaviors available that GSMs depend on but there are no manager specific messages or related behaviors that GSMs must support.

Every IBA compliant channel adapter, switch, or router shall support the functionality characterized as an SMA. Supporting the functionality characterized as a SMA means that the channel adapter, switch, or router sources and sinks messages and effects related behavior as specified in the corresponding class specific section of Chapter 16: General Services on page 748. The specific requirements for supporting this functionality at the various ports of the device are specified in the chapter covering the specific type of device. See Chapter 17: Channel Adapters on page 822, Chapter 18: Switches on page 845 and Chapter 19: Routers on page 862.

Every IBA compliant channel adapter, switch, or router supports the functionality characterized as the various GSAs for those general services specified to be mandatory in the class specific section of Chapter 16: General Services on page 748. Supporting the functionality characterized as a GSA means that the channel adapter, switch, or router sources and sinks messages and effects related behavior as specified in the corresponding class specific section of Chapter 16: General Services on page 748. The specific requirements for supporting this functionality at the various ports of the device are specified in the chapter covering the specific type of device. See Chapter 17: Channel Adapters on page 822, Chapter 18: Switches on page 845 and Chapter 19: Routers on page 862.

13.4 MANAGEMENT DATAGRAMS

Management Datagrams (MADs) are the basic elements of the messaging scheme defined for management communications. MADs are classified into predefined management classes and for each MAD there is a specified format, use, and behavior. This section specifies characteristics, i.e. formats and associated behaviors, common to all MADS or common across multiple classes. MADs specific to a class are specified in class specific sections of Chapter 14: on page 641, Chapter 15: on page 701, and Chapter 16: on page 748.

13.4.1 CONVENTIONS

C13-2: For all MADs, for both the fields in the MAD header as well as the fields in MAD attributes, bit placement follows the conventions specified in 1.5 Document Conventions on page 44. In addition, the following conventions shall be observed.

- Fields within a MAD may be either fixed length or variable length within a fixed length location. A variable length field placed in a fixed length location is placed in the high order bits of the fixed length location and the remainder of that location is filled with zero.
- Reserved fields must be filled with 0 by the requester and ignored by the receiver.
- When constructing a response MAD that contains all or part of the corresponding request MAD, it is acceptable to include the contents of reserved fields in the request MAD in the response MAD without regard to their content. That is, such fields need not be set to zero in the response MAD.
- In attribute descriptions in subsequent sections, fields specified as read only (RO) are not alterable by means of MADs. The mechanisms for setting such fields are implementation dependent and outside of the scope of the IBA. With respect to MADs that set values, recipients shall ignore any bits in the attribute in a request that correspond to RO components of the attribute being set.
- In attribute descriptions in subsequent sections, fields specified as read write (RW) are settable by means of MADs.

13.4.2 MANAGEMENT DATAGRAM FORMAT

C13-3: The data payload (as used in Chapter 9: Transport Layer on page 203) for all MADs shall be exactly 256 bytes.

C13-4: The data payload shall include, and only include, the items defined in the MAD base format in Figure 143 MAD Base Format on page 605, with semantics as described in Table 99 Common MAD Fields on page 605.

All MADs consist of a MAD header and MAD data. Except as noted, the MAD header definition is the same for all MADs. The contents of MAD data areas vary by management class and the specific attribute within the class.

Figure 143 MAD Base Format

bytes					
0	BaseVersion	MgmtClass	ClassVersion	R	Method
4	Status		ClassSpecific		
8	TransactionID				
12					
16	AttributeID		Reserved		
20	AttributeModifier				
24	Data				
...					
252					

13.4.3 MANAGEMENT DATAGRAM FIELDS

Table 99 Common MAD Fields on page 605 lists fields that are common to all MADs. Each class may specify additional class specific usage for certain of these fields.

Table 99 Common MAD Fields

Field Name	Length (bits)	Offset (bits)	Description
BaseVersion	8	0	Version of MAD base format. This shall be 1.
MgmtClass	8	8	Class of operation. See Table 100 Management Class Values on page 606 for definition and use.
ClassVersion	8	16	Version of MAD class-specific format. This shall be 1, except for the Vendor class where it shall be 1 or greater subject to vendor versioning.
R	1	24	Response bit. See 13.4.5 Management Class Methods on page 607 for definition and usage.
Method	7	25	Method to perform based on the management class. See 13.4.5 Management Class Methods on page 607 for definition and usage.
Status	16	32	Code indicating status of operation. See 13.4.7 Status Field on page 617 for definition and usage.
ClassSpecific	16	48	This field is reserved except for the Subnet Management class. See 14.2.1.2 SMP Data Format - Directed Route on page 643 for definition and usage for Subnet Management.

Table 99 Common MAD Fields

Field Name	Length (bits)	Offset (bits)	Description
TransactionID	64	64	Transaction identifier. See 13.4.6.4 TransactionID usage on page 616. This field, if unused by the management class, shall be set to 0.
AttributeID	16	128	Defines objects being operated on by a management class. This field, if unused, shall be set to 0. See 13.4.8 Management Class Attributes on page 617 as well as class specific sections of Chapter 14: on page 641, Chapter 15: on page 701, and Chapter 16: on page 748 for definition and usage.
Reserved	16	144	
AttributeModifier	32	160	Provides further scope to the attributes. Usage is determined by the management class and attribute. This field, when not used for the combination of management class and attribute specified in the header, shall be set to 0.
Data	1856	192	The data area, usage is defined within the scope of the management class.

13.4.4 MANAGEMENT CLASSES

C13-5: MADHeader:MgmtClass shall be one of the values defined in Table 100 Management Class Values on page 606 not defined as reserved.

The functionality provided by specific classes is specified in Chapter 14: Subnet Management on page 641, Chapter 15: Subnet Administration on page 701, and Chapter 16: General Services on page 748.

Table 100 Management Class Values

Management Class	Value	Description	Required Support for Class	Reference Section
Subn	0x01	Subnet Management class (LID routed)	All channel adapters, switches, and routers.	14.2 Subnet Management Class on page 642
Subn	0x81	Subnet Management class (Directed route)	All channel adapters, switches, and routers.	14.2 Subnet Management Class on page 642
SubnAdm	0x03	Subnet Administration class	All channel adapters, switches, or routers, hosting a subnet manager	15.2 SA MADs on page 702
Perf	0x04	Performance Management class	All channel adapters, switches, and routers.	16.1 Performance Management on page 748

Table 100 Management Class Values

Management Class	Value	Description	Required Support for Class	Reference Section
BM	0x05	Baseboard Management class (tunneling of IB-ML commands through the IBA subnet)	All channel adapters, switches, and routers.	16.2 Baseboard Management on page 781.
DevMgt	0x06	Device Management class	Optional.	16.3 Device Management on page 793
CommMgt	0x07	Communication Management class	All channel adapters that support RC,UC or RD.	16.7 Communication Management on page 818
SNMP	0x08	SNMP Tunneling class (tunneling of the SNMP protocol through the IBA fabric)	Optional	16.4 SNMP Tunneling on page 804
Vendor	0x09-0x0F	Vendor Specific classes	Optional	16.5 Vendor-specific on page 813
Application	0x10-0x1F	Application Specific classes	Optional	16.6 Application-specific on page 815
	0x00 0x20-0x80 0x82-0xFF	Reserved		

With respect to the column labeled Required Support for Class, an indication that support is required indicates that at least some aspects of the class must be supported. Complete details of which aspects are mandatory and which aspects are optional are specified in the corresponding reference section.

13.4.5 MANAGEMENT CLASS METHODS

Methods define the operations that a management class supports. In addition to supporting methods common to multiple classes, each management class may define additional class specific methods.

The upper bit of the Method field is designated as the response bit (R). It is used to distinguish three types of messages based upon the type of method included in the header as follows:

- Message methods are methods for which no response is ever generated. The R bit is not set (i.e. it is 0) and the corresponding method with the R bit set is reserved and not used.
- Request methods are methods for which a response may be generated. The R bit is not set (i.e. it is 0) and the corresponding method with the R bit set is defined and potentially used to convey a response.
- Response methods are methods generated in response to receipt of a request method. The R bit is set (i.e. it is 1) and the corresponding method with the R bit not set is defined and used to trigger (request) the response.

See section 13.4.6 Management Messaging on page 609 for required request/response behavior.

C13-6: The method names and method values shown in Table 101 Common Management Methods on page 608 shall be used in a manner consistent with the descriptions contained in this subsection (13.4.4 Management Classes on page 606).

C13-7: The values assigned to the common methods shall not be used for any class-dependent method even if the common method is not supported.

C13-8: Class specific methods defined to be requests and responses shall conform to the request response definitions in this section, the request response requirements specified in Section 13.4.6.4 TransactionID usage on page 616, and shall use the R bit according to the semantics of types of methods defined above.

Table 101 Common Management Methods

Name	Type	Value (including R bit)	Description
Get()	Request	0x01	Request (read) an attribute from a channel adapter, switch, or router. See 13.4.6.1.1 Get/GetResp on page 610.
Set()	Request	0x02	Request a set (write) of an attribute in a channel adapter, switch, or router. See 13.4.6.1.2 Set/GetResp on page 610.
GetResp()	Response	0x81	The response from an attribute Get or Set request. See 13.4.6.1.1 Get/GetResp on page 610 and 13.4.6.1.2 Set/GetResp on page 610.
Send()	Message	0x03	Send a datagram. Does not require a response. See 13.4.6.1.3 Send on page 611.

Table 101 Common Management Methods

Name	Type	Value (including R bit)	Description
Trap()	Message	0x05	An unsolicited datagram sent from a channel adapter, switch, or router indicating an event occurred that may be of interest. See 13.4.6.1.4 Trap on page 612 and 13.4.9 Traps on page 624.
Report()	Request	0x06	Used to forward an event/trap/notice to interested party. See 13.4.6.1.6 Report/ReportResp on page 612 and 13.4.11 Event Forwarding on page 627.
ReportResp()	Response	0x86	Response to a Report(). See 13.4.6.1.6 Report/ReportResp on page 612 and 13.4.11 Event Forwarding on page 627.
TrapRepress()	Message	0x07	Instruct a trap sender to cease sending a repeated trap. See 13.4.6.1.5 TrapRepress on page 612 and 13.4.9 Traps on page 624 for usage.
		0x00, 0x04, 0x08-0x0F, 0x80, 0x82-0x85, 0x87-0x8F	Reserved.
		0x10-0x7F, 0x90-0xFF	Class-specific methods. Use is defined by the class.

For Get(), Set(), GetResp(), and Send() methods, the combinations of method and attribute that are valid are class specific and are specified in the respective class sections in Chapter 14: on page 641, Chapter 15: on page 701, and Chapter 16: on page 748.

For Trap() and TrapRepress(), Report(), and ReportResp() methods, attribute usage is specified in Sections 13.4.9 Traps on page 624, and 13.4.11 Event Forwarding on page 627.

13.4.6 MANAGEMENT MESSAGING

13.4.6.1 METHODS AND MESSAGE SEQUENCING

For each MAD received by a responder, a given method may require a response to be returned.

Responders generate responses as appropriate and required for each request MAD received as specified in sections 13.4.6.1.1 Get/GetResp on page 610, 13.4.6.1.2 Set/GetResp on page 610, and 13.4.6.1.6 Report/ReportResp on page 612

C13-9: Responders shall not coalesce responses.

The subsequent ladder diagrams illustrate management request / response behavior for valid MADs. The operations defined below assume the receipt of a valid MAD. A MAD is valid if it satisfies all applicable validation checks as specified in Section 13.5.3 MAD Validation on page 635.

13.4.6.1.1 GET/GETRESP

Get() requests the read of an attribute from a channel adapter, switch, or router.

C13-10: In response to a valid Get(), the responder shall generate a GetResp() which consists of one or more response MADs - the number is a function of class-specific requirements for the requested attribute.

The attribute contained in the GetResp() is determined according to the specific MADHeader:MgmtClass and MADHeader:AttributeID in the request

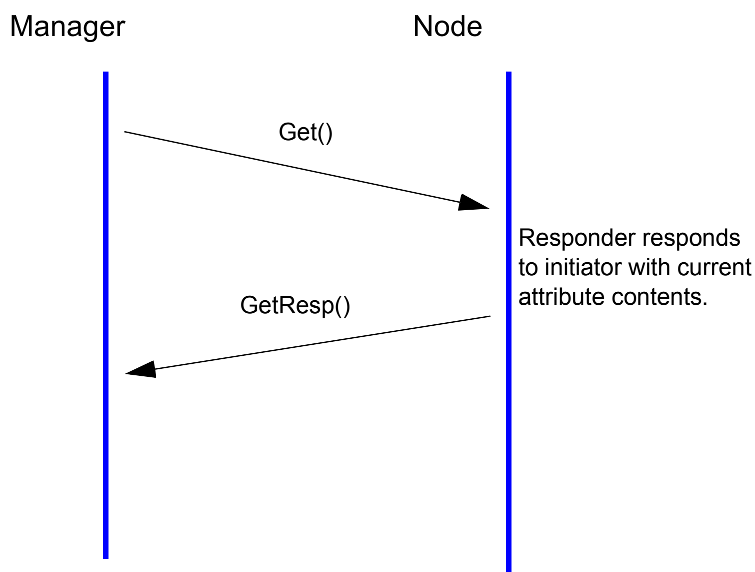


Figure 144 Get

13.4.6.1.2 SET/GETRESP

Set() informs the recipient to set values maintained by the recipient according to the values contained in the attribute conveyed in MADHeader:Data.

C13-11: In response to a valid Set(), the responder shall generate a GetResp() which consists of one or more response MADs - the number is a function of class-specific requirements for the requested attribute.[1]

C13-12: The attribute contained in the GetResp() shall be the same as in the associated request except as specified in C13-43:. The components of the attribute set shall be set equal to the equivalent values maintained

1. Exception: A Set() of PortInfo:PortPhysicalState to Disabled does not require that a GetResp() be sent from the port which has been Disabled.

by the recipient after the set has been performed except where otherwise stated in class specific sections.

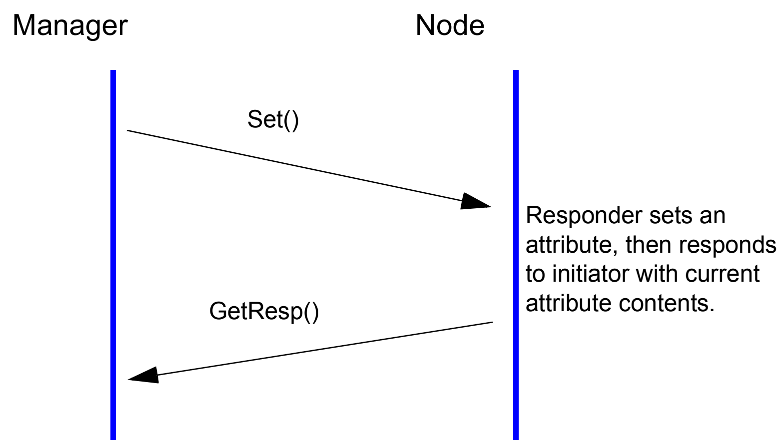


Figure 145 Set

13.4.6.1.3 SEND

Send() sends data from one entity to another on a class specific basis. If the class specific operations require reliability on top of the unreliable datagram service, higher level protocols may be defined based upon exchanges of send type MADs. Such higher level protocols are class specific and are defined either in the class specific sections or other sections referred to therein.

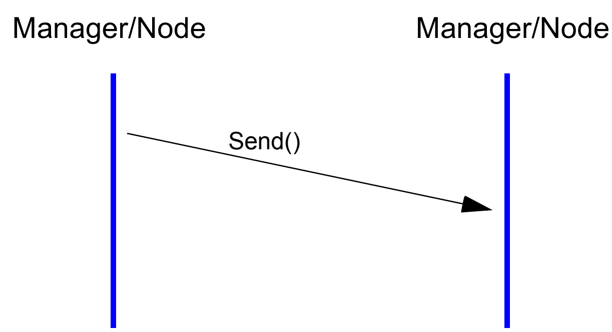


Figure 146 Send

13.4.6.1.4 TRAP

Trap() indicates an event occurred at a channel adapter, switch, or router. See Section 13.4.9 Traps on page 624 for the specification of trap usage and behavior.

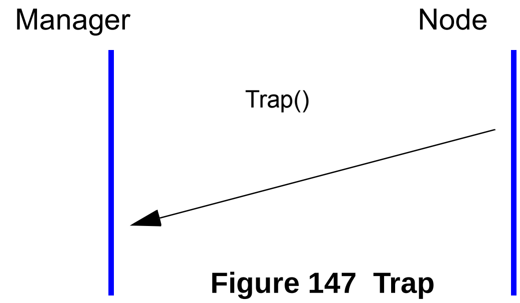


Figure 147 Trap

13.4.6.1.5 TRAPREPRESS

TrapRepress instructs a trap sender to cease sending a trap it is currently sending. See Section 13.4.9 Traps on page 624 for a complete specification of traps and TrapRepress.

The intended usage of TrapRepress() is shown below.

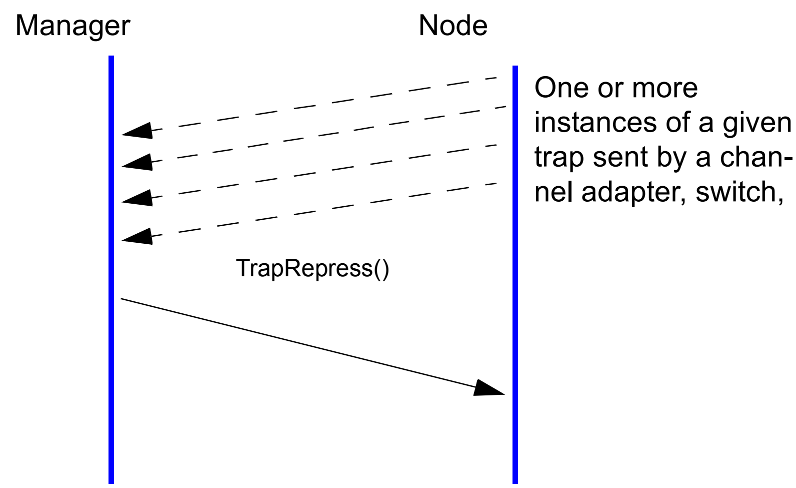


Figure 148 Trap

13.4.6.1.6 REPORT/REPORTRESP

Report() and ReportResp() MADs are used to forward traps directed to a GSM to parties who have subscribed for trap forwarding. The forwarding service is not available for subnet management class traps. See Section

13.4.11 Event Forwarding on page 627 for the complete specification of the event forwarding mechanism.

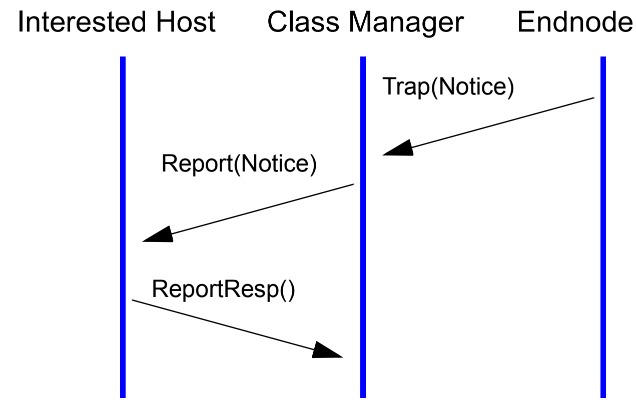


Figure 149 Forwarding traps/notices from the class manager

13.4.6.2 TIMERS AND TIMEOUTS

A management entity may use the IBA-defined management timeout and response time values to bound the amount of time a requester waits for a response or to bound the amount of time between successive MADs in a multiple MAD request, response, or message.

13.4.6.2.1 PORTINFO:SUBNETTIMEOUT

PortInfo:SubnetTimeout specifies the maximum expected propagation delay, which depends upon the configuration of the switches, to reach any other port in the subnet from the port with which this instance of PortInfo is associated. Requestors may use this value along with the appropriate RespTimeValue (below), to determine how long to wait for a response to a request before taking other action.

The duration of time is calculated as

4.096 microseconds * 2^PortInfo:SubnetTimeout.

Traps are subject to maximum trap rate injection constraints based upon PortInfo:SubnetTimeout. See 13.4.9 Traps on page 624 for the usage of PortInfo:SubnetTimout with respect to traps.

13.4.6.2.2 RESPTIMEVALUE

The IBA defined RespTimeValue specifies the expected maximum time interval between reception of an MAD and transmission of the associated response or between the associated port's transmission of successive MADs that are part of a multiple MAD sequence. Requestors may use this value along with the appropriate SubnetTimeout (above), to determine how long to wait for a response to a request, or, how long to wait for a succeeding MAD in a multi MAD sequence, as described in Section 13.4.6.3 Timeout/Timer Usage on page 615. The duration of time is calculated as

4.096 microseconds * 2^RespTimeValue.

C13-13: The default RespTimeValue shall be 8.

C13-14: The RespTimeValue applicable to a given situation depends upon the operation being performed and the MAD sequences involved. The appropriate RespTimeValue shall be determined as follows:

- If MADHeader:MgmtClass is Subn or Directed Route Subn the applicable RespTimeValue is conveyed by PortInfo:RespTimeValue (see 14.2.5.6 PortInfo on page 665 for the definition of PortInfo) of the relevant port as identified below.
- If MADHeader:MgmtClass is any other than Subn or Directed Route Subn, and the MADHeader:Method is not ReportResp(), the applicable RespTimeValue is conveyed by ClassPortInfo:RespTimeValue (see 13.4.8.1 ClassPortInfo on page 619 for the definition of ClassPortInfo) of the relevant port as identified below.
- If the MADHeader:Method is ReportResp(), the applicable RespTimeValue is conveyed by InformInfo:RespTimeValue (see Section 13.4.8.3 InformInfo on page 623 for the definition of InformInfo) specified by an event subscriber at the time of subscription.

C13-15: In the case of MAD sequences other than Report(), Report-Resp(), the port used to determine the applicable RespTimeValue shall be determined as follows:

- For MAD request-response exchanges consisting of a single packet request followed by a single packet response, the applicable RespTimeValue associated with the responding port indicates the expected maximum interval between receipt of the request at that port and initiation of transmission of the corresponding response.
- For MAD request-response exchanges including a multipacket request sequence followed by a response, the applicable RespTimeValue associated with the sending port indicates the expected maximum interval between initiation of transmission of successive packets in the multipacket request sequence. The applicable RespTimeValue associated with the receiving port indicates the expected maximum interval between receipt of the last packet of the multipacket request sequence at that port and initiation of transmission of the corresponding response.
- For MAD request-response exhanges including a multipacket response, the applicable RespTimeValue associated with the responding port indicates the expected maximum interval between receipt of the last packet of the request at that port and initiation of transmission of the response. The applicable RespTimeValue associated with the

responding port also indicates the expected maximum interval between the initiation of transmission of successive packets in the multi-packet response sequence.

- For MAD request-response exchanges using the windowing protocol defined in 15.3.3 Reliable Multi-packet Protocol Description on page 732, the applicable RespTimeValue associated with the port originating the request indicates the expected maximum time within which the requester will request more packets following the last packet in a burst of response packets.
- For operations requiring transmission of a sequence of multiple MADs not classified as requests (e.g. a succession of Send()s conveying fragments of an SNMP frame), the applicable RespTimeValue associated with the port sending the sequence indicates the expected maximum interval between initiation of transmission of successive packets in the sequence.

Send(), Trap(), or TrapRepress() do not have an associated response MAD (Send() MADs exchanged as part of a higher level protocol are not request/response sequences in this context). As such, the IBA-defined management timeout and response times are not applicable. Note that while TrapRepress() may be sent as a result of the sending of a trap, Trap() and TrapRepress() are classified as messages not as requests or responses and do not constitute a request/response sequence.

13.4.6.3 TIMEOUT/TIMER USAGE

In general, the expected maximum time interval between transmission of a request and receipt of the associated response is

2*PortInfo:SubnetTimeout + RespTimeValue

where RespTimeValue is determined according to Section 13.4.6.2.2 RespTimeValue on page 613.

In general, the expected maximum time interval between reception of successive MADs in a multi MAD transfer is

PortInfo:SubnetTimeout + RespTimeValue

where RespTimeValue is determined according to Section 13.4.6.2.2 RespTimeValue on page 613.

If either expected maximum time interval is exceeded, the recipient of the MADs may consider the entire sequence invalid. The exact behavior in this case may be class-specific and possibly attribute-specific, but the recipient can always reclaim the resources used by the part of the sequence already received, and discard any MADs in the sequence that arrive later. In the case of the reliable transport protocol (15.3.3 Reliable Multi-packet

Protocol Description on page 732) retransmission of a subsequence may be requested.

C13-16: For request/response sequences, timers shall be started for each request transmitted and reset upon arrival of the corresponding response MAD.

C13-17: For multi MAD sequences, timers shall be reset then started upon arrival of each successive MAD in the sequence

13.4.6.4 TRANSACTIONID USAGE

The contents of the TransactionID (TID) field are implementation-dependent.

C13-18: When initiating a new operation, MADHeader:TransactionID shall be set to such a value that within that MAD the combination of TID, SGID, and MgmtClass is different from that of any other currently executing operation. Repeated Trap messages for the same event may be regarded as continuing a 'currently executing' operation as long as the trap can be repeated and no corresponding TrapRepress has been received, or they may be regarded as initiating new operations.

C13-19: Note that the above implies that recipients of messages shall use the combination of TID, SGID, and MgmtClass to uniquely associate messages or message sequences, not just the TID.

C13-20: When constructing a request that consists of sequence of MADs, requesters shall set MADHeader:TransactionID in each MAD that is part of the sequence to an identical value.

C13-21: When constructing a response, responders shall set MADHeader:TransactionID in the response equal to MADHeader:TransactionID in the corresponding request.

C13-22: Where a response is made up of multiple MADs, MADHeader:TransactionID in each MAD in the response shall be set equal to MADHeader:TransactionID in the corresponding request.

C13-23: Where an operation defined for an IBA management class requires a sender to send a succession of MADs of type message to effect the operation (e.g. an SNMP PDU being tunneled through IBA), the sender shall set MADHeader:TransactionID in each MAD that is part of the sequence to an identical value.

13.4.7 STATUS FIELD

All MADs contain a status field. The status field is used in MADs of type response to convey information about the disposition of the request or conditions associated with disposition of the request.

The status field consists of 16 bits. The eight low order bits of the field are used for indications common to all classes. The eight high order bits of the field are used for class specific indications. Class specific status indications are defined in the class specific sections of Chapter 14: Subnet Management on page 641, Chapter 15: Subnet Administration on page 701, and Chapter 16: General Services on page 748.

C13-24: For messages of type Response (see 13.4.5 Management Class Methods on page 607), the usage of the low order 8 bits shall be set as specified in Table 102 MAD Common Status Field Bit Values on page 617.

Table 102 MAD Common Status Field Bit Values

Name	Bit	Meaning
0	Busy	Temporarily busy. MAD discarded. This is not an error.
1	Redirect_required	Redirection. This is not an error.
2-4	Code for invalid field	0 - no invalid fields 1 - The class version specified is not supported. 2 - The method specified is not supported 3 - The method/attribute combination is not supported 4-6: Reserved 7 - One or more fields in the attribute contain an invalid value.
5-7	Reserved.	
8-15	Class Specific	The use of these bits is class specific.

C13-25: For messages of type Request or type Message (see Section 13.4.5 Management Class Methods on page 607), the entire status field shall be set to 0.

13.4.8 MANAGEMENT CLASS ATTRIBUTES

Attributes define the data which a management class manipulates. Each management class defines its own set of attributes.

Attributes are composite structures consisting of components typically representing hardware registers in channel adapters, switches, or routers. Each attribute is assigned a unique Attribute ID.

Depending upon the attribute, components may be read only, read/write, or reserved.

Some attributes have associated Attribute Modifiers (AMs) which further qualify or modify the application of the attribute. The use of the AM is attribute-specific and usage is defined where the attribute is defined.

C13-26: When the AM is not used it shall be set to all zeroes.

It is not possible to selectively set a single component within an attribute. A Get() must be performed to obtain the whole attribute, the single component must be modified in the result and a Set() must be performed to write the whole attribute. No atomicity is implied or provided in this sequence of operations.

C13-27: A given attribute shall have the same format for the Get(), Set() and GetResp() methods if used with those methods.

There are three attributes which are common across multiple classes. Table 103 Attributes Common to Multiple Classes on page 618 lists each such attribute, its ID, and the classes where it is used. Attribute IDs less than 0x10 identify common attributes or are reserved. Attribute IDs equal to or greater than 0x10 identify attributes whose definitions are class specific. The structure and content of the common attributes is defined in the following subsections. The structure and content of class specific attributes are defined in the respective class specific sections of Chapter 14: Subnet Management on page 641, Chapter 15: Subnet Administration on page 701, and Chapter 16: General Services on page 748.

The following common attributes are defined:

Table 103 Attributes Common to Multiple Classes

Attribute Name	Attribute ID	Attribute Modifier	Description	Where Used
	0x0000		Reserved	
ClassPortInfo	0x0001	0x00000000	General and port-specific information for a GS management class	The SA class and all supported GS classes on channel adapters, switches, and routers (See 13.4.8.1 ClassPortInfo on page 619).
Notice	0x0002	0x00000000-0xFFFFFFFF	Information regarding the associated Notice (or Trap in which case the Attribute Modifier shall be 0)	All classes supporting traps/notices. (See 13.4.8.2 Notice on page 622).

Table 103 Attributes Common to Multiple Classes

Attribute Name	Attribute ID	Attribute Modifier	Description	Where Used
InformInfo	0x0003	0x00000000	Event Subscription	All classes having a class manager supporting event subscription. (See 13.4.8.3 InformInfo on page 623).
	0x0004-0x000F		Reserved	
	0x0010-0xFFFF		Class-dependent values.	Usage of values in this range is class specific and is specified in the class specific sections of Chapter 14: on page 641, Chapter 15: on page 701, and Chapter 16: on page 748.

13.4.8.1 ClassPortInfo

C13-28: Channel adapters, switches, and routers implementing a GS class shall implement ClassPortInfo according to the definition specified in Table 104 ClassPortInfo on page 620.

C13-29: The ClassPortInfo attribute shall be implemented for every GS class supported by a node; it shall be implemented on every port through which the GS class may be accessed.

C13-30: The ClassPortInfo attribute shall be implemented for the SA class by any node on which an SA is located; it shall be implemented on every port through which the SA class may be accessed.

The presence of ClassPortInfo for a management class confirms the availability of that management class on a particular channel adapter, switch, or router and provides information about the version of MADs supported by the class on that channel adapter, switch, or router. (Note: support for a given class can be determined directly from capability bits in PortInfo for the port in question. See 14.2.5.6 PortInfo on page 665).

The ClassPortInfo attribute also provides port-specific information for class services on a channel adapter, switch, or router. In addition to being available as the object of a Get() method specifying it as the target, ClassPortInfo is also returned as the result of any Get() or Set() if the requester is being redirected as described in Section 13.5.2 GSI Redirection on page 634.

ClassPortInfo contains information related to general services traps. If sending trap messages is supported on a channel adapter, switch, or router, and if trap sending is enabled for this port (nonzero TrapLID), ClassPortInfo defines the destination to which traps for the subject GS class applying to this port are to be sent. See Section 13.4.9 Traps on page 624. Note that this applies only to general services traps. Subnet management traps do not use this mechanism.

For both redirection and traps, ClassPortInfo provides support for cross-subnet communications by including the information necessary to build a properly formed GRH, see Sections 13.4.9 Traps on page 624 and 13.5.2 GSI Redirection on page 634.

Table 104 ClassPortInfo

Component	Access	Length (bits)	Offset (bits)	Description
BaseVersion	RO	8	0	Current supported MAD Base Version. Indicates that this channel adapter, switch, or router supports up to and including this version.
ClassVersion	RO	8	8	Current supported management class version. Indicates that this channel adapter, switch, or router supports up to and including this version.
CapabilityMask	RO	16	16	Supported capabilities of this management class, bit set to 1 for affirmation of management support. Bit 0 - If 1, the management class generates Trap() MADs Bit 1 - If 1, the management class implements Get(Notice) and Set(Notice) Bit 2-7: reserved Bit 8-15: class-specific capabilities.
Reserved	RO	27	32	Reserved
RespTimeValue	RO	5	59	See 13.4.6.2 Timers and Timeouts on page 613.
RedirectGID	RO	128	64	The GID a requester shall use as the destination GID in the GRH of messages used to access redirected class services. If redirection is not being performed, this shall be set to zero.
RedirectTC	RO	8	192	The Traffic Class a requester shall use in the GRH of messages used to access redirected class services. For more on the definition and significance of traffic class see 8.2.2.3 Service Levels on page 196 and 8.3.2 Traffic Class (TClass) - 8 bits on page 198
RedirectSL	RO	4	200	The SL a requester shall use to access the class services.
RedirectFL	RO	20	204	The Flow Label a requester shall use in the GRH of messages used to access redirected class services.

Table 104 ClassPortInfo

Component	Access	Length (bits)	Offset (bits)	Description
RedirectLID	RO	16	224	If this value is non-zero, it is the DLID a requester shall use to access the class services. If this value is zero, the redirect requires the requester to use the supplied RedirectGID to request further path resolution from subnet administration. The RedirectGID, the RedirectQP and RedirectP_Key from this redirect response are all valid, but the RedirectSL, RedirectFL, RedirectTC, and RedirectLID will in general not be valid; they must be replaced using a PathRecord obtained from the SA. See the comment about redirection following this table.
RedirectP_Key	RO	16	240	The P_Key a requester shall use to access the class services.
Reserved	RO	8	256	Reserved
RedirectQP	RO	24	264	The QP a requester shall use to access the class services. Zero is illegal.
RedirectQ_Key	RO	32	288	The Q_Key associated with the RedirectQP. This Q_Key shall be set to the well known Q_Key.
TrapGID	RW	128	320	The GID to be used as the destination GID in the GRH of trap messages originated by this service. If all zeroes, no GRH is inserted in trap messages.
TrapTC	RW	8	448	The Traffic Class to be placed in the GRH of trap messages originated by this service. For more on the definition and significance of traffic class see 8.2.2.3 Service Levels on page 196 and 8.3.2 Traffic Class (TClass) - 8 bits on page 198.
TrapSL	RW	4	456	The SL that shall be used when sending trap messages originated by this service.
TrapFL	RW	20	460	The Flow Label to be placed in the GRH of trap messages originated by this service.
TrapLID	RW	16	480	The DLID to where trap messages shall be sent by this service. If all zeroes, traps shall not be sent from this port.
TrapP_Key	RW	16	496	The P_Key to be placed in the header for traps originated by this service.
TrapHL	RW	8	512	The Hop Limit to be placed in the GRH of trap messages originated by this service. This specifies the maximum number of routers through which the message containing the GRH specified here may pass. The default value is 255.
TrapQP	RW	24	520	The QP to which trap messages originated by this service traps shall be sent. Must not be zero.
TrapQ_Key	RW	32	544	The Q_Key associated with the TrapQP. This Q_Key shall have the high order bit set. See 10.2.4 Q_Keys on page 403 for a description of the significance of setting the high order bit.

Comment About Redirection: Redirection may be fairly complex on certain fabric topologies. Simple InfiniBand CAs may not be able to fully resolve a path to another port it may supply, while complex InfiniBand management applications may do this as a matter of course. In the former case the simple CA can send a redirect to a requester that sets the RedirectLID component in the ClassPortInfo Attribute to zero. A zero value RedirectLID indicates that the requester must request a PathRecord from the SA using the supplied RedirectGID and requester's own source information. Refer to 15.2.5.17 PathRecord on page 717.

13.4.8.2 NOTICE

The Notice attribute describes an exception or other channel adapter, switch, or router event. It is used by both the trap mechanism described in 13.4.9 Traps on page 624 and the Notice mechanism described in 13.4.10 Notice Queue on page 626.

o13-1: Channel adapters, switches, and routers implementing Notice attributes shall conform to the definition specified in Table 105 Notice on page 622.

Table 105 Notice

Component	Access	Length (bits)	Offset (bits)	Description
IsGeneric	RO	1	0	If set to 1, notice is generic, else is vendor specific
Type	RO	7	1	Enumeration indicating type of trap/notice: 0 - Fatal 1 - Urgent 2 - Security 3 - Subnet Management 4 - Informational 5-0x7E - Reserved 0x7F - Empty notice. All other fields are meaningless.
NodeType / VendorID	RO	24	8	If generic, indicates Node Type: 1 - Channel Adapter 2 - Switch 3 - Router 4 - Subnet Management 0, 5-0xFFFFFF - Reserved If not generic, indicates the 24 bit IEEE OUI assigned to the vendor.
TrapNumber / DeviceID	RO	16	32	If generic, indicates a class-defined trap number. Number 0xFFFF is reserved. If not generic, this is Device ID information as assigned by device manufacturer.

Table 105 Notice

Component	Access	Length (bits)	Offset (bits)	Description
IssuerLID	RO	16	48	LID of issuer.
NoticeToggle	RO	1	64	For Notices, alternates between zero and one after each Notice is cleared. See Section 13.4.10 Notice Queue on page 626. For Traps, this shall be set to 0.
NoticeCount	RO	15	65	For Notices, indicates the number of notices queued on this channel adapter, switch, or router. See Section 13.4.10 Notice Queue on page 626. For Traps, this shall be set to all zeroes.
DataDetails	RO	432	80	If generic, data details is disambiguated by management class and TrapNumber. Otherwise disambiguation is vendor defined.

Certain operations involving the Notice attribute require the use of an empty notice. See 13.4.10 Notice Queue on page 626 and 13.4.11 Event Forwarding on page 627. An empty notice is a Notice attribute in which the type field is set to 0x7F. The only valid field in an empty notice is the type field, the contents of all others should be considered meaningless.

13.4.8.3 InformInfo

The InformInfo attribute provides information for subscribing to a class manager for event forwarding. See13.4.11 Event Forwarding on page 627.

o13-2: Channel adapters, switches, and routers implementing the InformInfo attribute shall conform to the definition specified in Table 106 InformInfo on page 623.

Table 106 InformInfo

Component	Type	Length (bits)	Offset (bits)	Description
GID	RW	128	0	Specifies specific GID to subscribe for. Set to all zeroes if not desired. See Section 13.4.11 Event Forwarding on page 627.
LIDRangeBegin	RW	16	128	Specifies the lowest LID in a range of LID addresses to subscribe for. Address 0xFFFF denotes all LID addresses.
LIDRangeEnd	RW	16	144	Specifies the highest LID in a range of LID addresses to subscribe for. Set to 0 if no range desired. Ignored if LIDRangeBegin is 0xFFFF.
P_Key	RW	16	160	The partition key to use.

Table 106 InformInfo

Component	Type	Length (bits)	Offset (bits)	Description
IsGeneric	RW	8	176	If set to 1, forward generic traps. If set to 0, forward all vendor specific traps. Values above 1 are undefined.
Subscribe	RW	8	184	If set to 1, subscribe If set to 0, unsubscribe. Values above 1 are undefined.
ClassRange	RW	16	192	Enumeration indicating class of trap/notice. Valid values are: 0 - Fatal 1 - Urgent 2 - Security 3 - Subnet Management 4 - Informational 0xFFFF - forward all
DeviceID/TrapNumber	RW	16	208	If not generic, this is device ID information as assigned by device manufacturer. If generic, indicates trap number. Number 0xFFFF means forward any Device ID/ TrapNumber.
RespTimeValue	RO	32	224	See Section 13.4.6.2.2 RespTimeValue on page 613.
Reserved	RO	8	256	
VendorID / NodeType	RW	24	264	If generic, indicates Node Type: 1 - Channel Adapter 2 - Switch 3 - Router 4 - Subnet Management 0, 5-0xFFFFFF - Reserved If not generic, indicates the 24 bit IEEE OUI assigned to the vendor.

13.4.9 TRAPS

Traps are asynchronous notifications for the purpose of alerting an entity within another channel adapter, switch, or router about exception conditions or other events of interest at a given channel adapter, switch, or router within the subnet.

C13-31: Either Trap support, or Notice support, or both shall be provided by all managers.

See 13.4.10 Notice Queue on page 626 for a description of Notice support.

For the subnet management class, traps originate at a CA, switch, or router and are always sent to the master subnet manager managing the originator. The managing SM is always on the same subnet since cross subnet communications is not allowed for subnet management class MADs. Subnet management traps are always allowed and there is no direct mechanism for preventing a CA, switch, or router from sending traps (note, there may be actions which as a side effect cause cessation of traps, i.e. downing a port, but these are not considered direct mechanisms).

For each GS management class, whether traps are allowed to be sent for that class is specified in ClassPortInfo for that class.

C13-32: If ClassPortInfo:TrapLID for a particular port and class is zero, traps shall not be generated from that port for that class.

o13-2.a1: If trap generation is supported, the destination address and certain other necessary message parameters shall be obtained from ClassPortInfo as indicated in table 106 and the source LID shall be set to the PortInfo:LID of the originating port.

The destination for traps may be in the same subnet or in another subnet. If traps are to be delivered to a destination not in the same subnet, the ClassPortInfo:TrapGID is non zero and a GRH is required in the trap message. If ClassPortInfo:TrapGID is zero, the trap destination is within the same subnet and no GRH is included in the message.

If a GRH is required, the source GID in the GRH is the GIDIndex0 of the originating port and other fields in the GRH are either fixed or are given values drawn from counterpart fields in ClassPortInfo. 8.3 Global Route Header on page 198 specifies the requirements applicable to GRHs.

Traps are always sent to the destination identified in ClassPortInfo and only to that destination. It is the responsibility of any entity that sets ClassPortInfo fields to assure that the values programmed are consistent. That is, the combination of GID/DLID, P_Key, port, etc. must be consistent with addressing of and access rules applicable to the specified port.

Traps may be issued by any channel adapter, switch, or router on the subnet. Channel adapters, switches, and routers may repeat sending of a Trap()

o13-3: Channel Adapters, switches and routers shall not send traps at a rate greater than the trap rate limit specified. For a given port, the trap rate limit shall be defined as the reciprocal of the time duration determined from PortInfo:SubnetTimeout for that port. See Section 13.4.6.2.1 PortInfo:SubnetTimeout on page 613.

o13-4: Traps shall contain the Notice attribute to identify the trap. The Notice attribute is described in 13.4.8.2 Notice on page 622.

o13-5: Trap originators shall use the same MADHeader:TransactionID value for all instances of repeated traps.

Recipients of traps may send TrapRepress() MADs to trap originators.

o13-6: Upon receipt of a valid TrapRepress() MAD, the trap originator shall cease sending the trap which matches the trap identified by the TrapRepress() MAD. A trap being repeatedly sent matches a trap identified in a TrapRepress() MAD when both MADHeader:TransactionID in the trap MAD matches MADHeader:TransactionID in the TrapRepress MAD and the Notice attribute in the trap MAD matches the Notice attribute in the TrapRepress().

o13-7: If a TrapRepress() is received and no matching trap is being sent, the TrapRepress() shall be silently dropped and no other action taken.

Sending traps is optional, unless specified otherwise by a compliance statement.

Management classes may choose to supplement trap handling through the use of the event forwarding mechanism described in Section 13.4.11 Event Forwarding on page 627.

Although Traps and the Notice Queue (NQ) mechanism, Section 13.4.10 Notice Queue on page 626, use the same Notice attribute to describe events or conditions, the trap mechanism and the notice queues mechanism are completely independent. There is no requirement that a Notice queue entry be generated when a Trap is sent or vice versa.

13.4.10 NOTICE QUEUE

The NQ is a repository for storing Notice attributes (see 13.4.8.2 Notice on page 622) associated with the occurrence of an event or the detection of a condition at a channel adapter, switch, or router. Notices in the NQ are queried or deleted using Get(Notice) and Set(Notice) methods respectively.

o13-8: The Notice Queue shall operate as a first in first out queue. For Get(Notice), the AM shall be 0. This selects the oldest notice attribute saved, that is, the Notice attribute on the top (or front) of the queue.

o13-9: Notice:NoticeCount in a returned Notice attribute shall always indicate the number of notices currently on the queue. Performing a Get(Notice) does not remove a Notice from the NQ. Notice:NoticeCount includes the notice returned in response to the Get().

o13-10: If the queue is empty, the Notice attribute returned shall be an empty Notice and Notice:NoticeCount shall contain 0. Otherwise the recipient of a Get(Notice) shall return a copy of the selected notice in the Notice attribute in the response unless the queue is empty.

To clear notices from the head of the Notice Queue, the requester sends a Set(Notice) MAD containing a Notice attribute with:

- Notice:NoticeToggle set to match Notice:NoticeToggle in the channel adapter's, switches', or router's Notice attribute.
- Notice:NoticeCount set to the number of notices to delete.
- MADHeader:AM = 0

o13-11: Upon receipt of a Set(Notice) MAD, if the recipient implements an NQ, the recipient shall perform the following actions:

- If the NoticeToggle value in the Set(Notice) does not match the NoticeToggle value in the Notice attribute on the channel adapter, switch, or router, the Set(Notice) is silently discarded and no other action is taken.
- The oldest notice and successively newer notices up to a total number of notices indicated by Notice:NoticeCount in the Notice attribute contained in the Set(Notice) shall be deleted. If Notice:NoticeCount in the request Notice is greater than the number of notices on the queue, the queue is emptied.
- The response to the next Get(Notice) request shall return a Notice attribute that corresponds to the new top of the queue and Notice:NoticeCount in the response shall reflect the updated count of notices on the queue.
- Since the Notice Queue acts as a FIFO, the only valid value for MADHeader:AM is 0.

The types and number of notices captured by a channel adapter, switch, or router is implementation-dependent. The actual size of the Notice Queue is implementation specific and is not specified by the architecture. Behavior of a full Notice Queue when the channel adapter, switch, or router has another notice to queue is undefined.

Channel adapters, switches, and routers are not required to support the NQ mechanism.

13.4.11 EVENT FORWARDING

Nodes can request that specific traps sent to a class manager by a given channel adapter, switch, or router be forwarded to them by subscribing for traps from channel adapter, switch, or router. This is done via the event-forward subscription mechanism. To subscribe, an interested host sends a Set(InformInfo) request to the class manager identifying the channel adapter, switch, or router it wishes traps to be forwarded from by its GID, LID, or identifying a set of channel adapters, switches, or routers whose LIDs are in a specified LID range. The class manager responds with a GetResp(InformInfo) message to confirm or deny such forwarding.

o13-12: A manager confirming a request for event subscription shall respond with InformInfo:Subscribe set to 1.

o13-13: A manager denying a request for event subscription shall respond with InformInfo:Subscribe set to 0.

Requestors wishing to subscribe to event forwarding may determine which managers exist and their locations on the fabric by querying the SA. See 15.4 Operations on page 737 for a discussion of subnet administration including restrictions on access to and use of the SA.

The exchange of MADs to effect subscription to event forwarding is depicted in Figure 150 Subscribing and unsubscribing for forwarding on page 628 below.

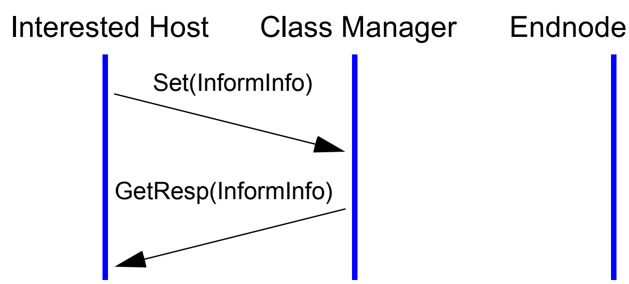


Figure 150 Subscribing and unsubscribing for forwarding

o13-14: Managers receiving a Set(InformInfo) shall verify that the requestor originating the Set(InformInfo) and a trap source identified by InformInfo:GID, InformInfo:LIDRangeBegin, or by a LID included in the range InformInfo:LIDRangeBegin-InformInfo:LIDRangeEnd are permitted to access each other according to the current partitioning. The manager shall perform verification by verifying that a valid path exists between the requestor and the trap source.

This verification can be accomplished by requesting a path between those two using a SA query operation. If such a path exists, the Set(InformInfo) succeeds. If such a path does not exist, the LID is invalid as a trap source. A LID specifies an invalid trap source if it is not assigned or if the associated port is not accessible to the requestor under current partitioning.

o13-15: If partition verification fails on Set(InformInfo), the manager receiving the request shall indicate in the response that the operation failed with an invalid attribute status value as defined in Section 13.4.7 Status Field on page 617 and Table 102 MAD Common Status Field Bit Values on page 617).

o13-16: If InformInfo:LIDRangeBegin-InformInfo:LIDRangeEnd specifies a range, all LIDs in the range shall be checked for validity as trap sources. If all are valid or invalid, operation shall be the same as if a single trap source was specified. If InformInfo:LIDRangeBegin-InformInfo:LIDRangeEnd specifies a range including one or more LIDs not valid as trap sources, the manager shall either indicate the entire request is invalid, using the invalid attribute status as above; or shall use a class-specific means to indicate that some of the LIDs were associated with valid trap sources and others were not, identifying the invalid ones using the invalid attribute status.

o13-17: Managers which have confirmed a request for event subscription shall forward corresponding events to the subscriber via the Report(Notice) MAD.

The recipient of a Report(Notice) datagram should respond with a ReportResp() MAD with the same transaction ID as issued by the class manager and an empty Notice attribute.

Figure 151 Forwarding traps/notices from the class manager on page 629, depicts the MAD exchange associated with forwarding traps to a subscriber. The report response provides means for the class manager to assure that subscribers receive reports assuming communications with the subscriber is not failed.

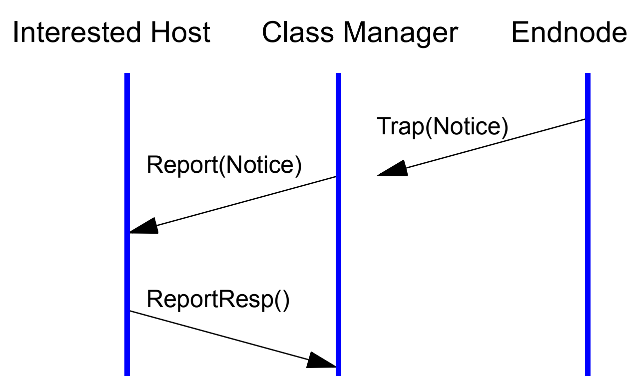


Figure 151 Forwarding traps/notices from the class manager

This forwarding service is not available directly from the Subnet Manager for Subnet Management traps. The SA must be used, see Chapter 15: Subnet Administration.

13.5 MAD PROCESSING

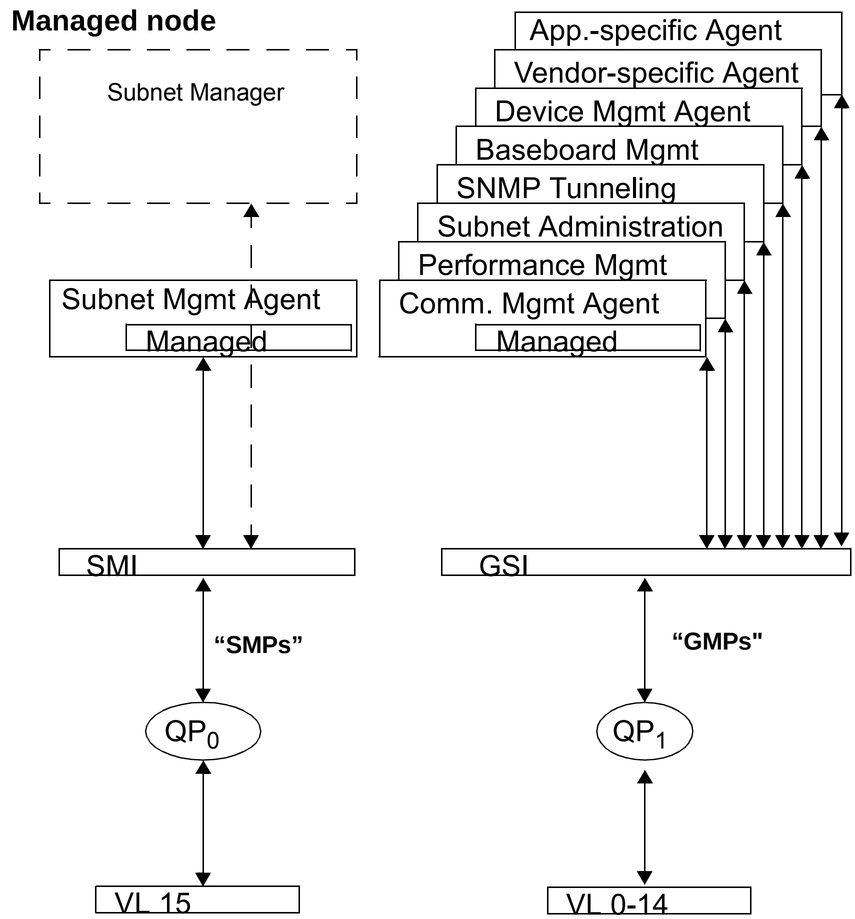
Non redirected MADs are distinguished from other packets by the destination queue pair specified in the packet. Two specific queue pair numbers are dedicated to supporting non redirected management operations. Each of the dedicated queue pairs represents a unique interface to one or more management services. These interfaces and the behaviors related to associated services are specified in subsequent sections.

If redirection is in effect, redirected MADs may be directed to a queue pair different from either of the dedicated queue pairs. How a management service is associated with such a queue pair is implementation specific. Such MADs are standard packets and MADs arriving at a port are directed according to the standard procedures for directing packets to queue pairs.

13.5.1 MAD INTERFACES

Two required interfaces to management entities are specified based upon two well known queue pairs. These are known as the Subnet Management Interface and the General Service Interface.

Figure 152 MAD Interface on page 631, below, depicts the general relationships among management entities and their interfaces to the wire. Note, the figure itself is meant to be representative of a basic channel adapter, switch, or router and is not meant to imply a specific implementation or to imply specific requirements or limitations.



Note: A channel adapter, switch, or router may or may not contain a subnet manager. If a CA, switch or router does contain a subnet manager, the specific relationships between the SM, the SMA, and the SMI are implementation specific.

Figure 152 MAD Interface

C13-33: For each endport, Subnet Management MADs to be processed at that port shall be destined to Queue Pair 0.

C13-34: For each endport, unless redirected (see Section 13.5.2 GSI Redirection on page 634), SA or GS MADs to be processed at that port shall be destined to Queue Pair 1.

Queue pairs 0 and 1 have unique semantics with respect to processing of messages specifying one of them as the destination queue pair. Implementations of QP 0 and 1 are not required to follow the semantics associated with other queue pairs with respect to requirements such as posting and consumption of WQEs, manipulation of an associated completion queue, and so on. Messages arriving at QP 0 or QP 1 are processed in accordance with the requirements set forth in this section and following Sections: 13.5.3.1 MAD validation for subnet management MADs on page 636, 13.5.3.2.1 MAD validation at the GSI on page 637, and 13.5.3.2.2 MAD validation at the SA and GSAs on page 638.

13.5.1.1 PROCESSING SUBNET MANAGEMENT PACKETS (SMPS)

The Subnet Management Interface (SMI) is associated with QP 0. QP 0 is used exclusively for sending and receiving subnet management MADs. Communications with the SMA in a channel adapter, switch, or router is always through the SMI. If a channel adapter, switch, or router hosts a SM, then communications between that SM and the SMA of each channel adapter, switch, or router in the subnet is also through the SMI. Only SMAs and SM communicate through this interface. No other entities may do so.

The MADs of subnet management class are called SMPs.

C13-35: SMPs shall not travel beyond the boundaries of a subnet (i.e. through a router).

MADs with a destination queue pair of 0 are validated according to the rules specified in section 13.5.3.1 MAD validation for subnet management MADs on page 636.

Validated MADs arriving for QP0 are handled by the SMI. It is not specified how the SMI dispatches the SMPs between the SMA and a possible SM.

C13-36: On an HCA, SMPs not dispatched to the SMA shall be posted to the QP0 queue pair exposed above the verb layer.

C13-37: For SMPs dispatched to the SMA, a vendor shall

- either never post such SMPs,
- or, always post such SMPs,
- or, offer a vendor specific option to select whether such SMPs are never posted or are always posted,

where posting is with respect to the QP0 queue pair exposed above the verb layer.

13.5.1.2 PROCESSING GENERAL SERVICES MANAGEMENT PACKETS (GMPS)

The General Services Interface (GSI) is associated with QP 1. QP 1 is reserved exclusively for subnet administration and general services MADs. Unless redirected, GSAs send and receive MADs by means of the GSI. For a description of redirection see 13.5.2 on page 634. Depending upon implementation, the GSI may also provide the interface through which a class manager communicates with corresponding (class specific) GSAs throughout the fabric.

The MADs defined for subnet administration and general services are referred to as GMPs. The GSI acts as a demultiplexor for GMPs, distributing messages destined for QP 1 to the appropriate service agent or class manager, based upon MADHeader:MgmtClass in the MAD header. MADs with a destination queue pair of 1 are validated according to the rules specified in section 13.5.3.2.1 MAD validation at the GSI on page 637.

In those cases where the GSI provides an interface for both a class service agent and the corresponding class manager, the determination of the appropriate destination above the GSI demultiplexing is implementation dependent.

On an HCA, the GSI is only aware of agents residing below the verb layer.

C13-38: GMPs dispatched to agents implemented below the verb layer shall not be visible above the verb layer.

C13-39: GMPs that are not dispatched to agents implemented below the verb layer shall be visible above the verb layer as posted to the QP1 exposed above the verbs.

The SL used by a GMP is neither specified nor constrained by virtue of the fact it is a GMP. The choice of SL is outside of the scope of these sections. Note that unlike SMPs which follow special and unique VL rules, GMPs are standard unreliable datagrams subject to and only to the SL/VL usage rules applicable to all unreliable datagrams.

If redirection has been configured for a management class, GMPs destined to the QP specified in the redirection are treated exactly the same as any other unreliable datagram. Since the destination QP is not QP 1, they do not appear at the GSI but are delivered directly to the QP specified in the redirection by the IB transport in the same manner as any other unreliable datagram.

C13-40: GSAs that are accessed using redirection shall validate arriving MADs according to the same rules as apply for queue pair 1.

GMPs may contain a GRH and may be forwarded across subnet boundaries. Whether or not a given class manager supports cross subnet communications with corresponding class service agents is implementation dependent.

Table 107 Management Interfaces Summary on page 634 summarizes the properties associated with the above described management interfaces.

Table 107 Management Interfaces Summary

	Subnet Management Interface	General Services Interface
Queue Pair	QP 0	QP 1
VL	VL 15	not VL 15
Partitioning	not enforced	enforced
Q_Key	not enforced	enforced (Q_Key = 0x8001_0000)
Scope	Within subnet only	Routable across subnets
Class Key	Management Key (M_Key)	class dependent

13.5.2 GSI REDIRECTION

By default, the interface from the wire to class service agents is the GSI. A mechanism is provided by which the interface to a given class service agent may be relocated to another queue pair. This mechanism is called redirection and is specified in detail below. The SA as well as each GSA may individually support this mechanism or not. The ClassPortInfo attribute is used to indicate if redirection is supported, and, if so, contains redirection information for MADs of the subject class.

C13-41: If, for a class, redirection is not being used, any GMP destined to the associated class agent via QP 1 shall be processed by that agent.

C13-42: For any request sent to QP 1 with MADHeader:MgmtClass equal to the class value of a class being redirected, a response shall be returned containing ClassPortInfo for the class specified in the request

C13-43: The Status field in a response including ClassPortInfo because of redirection shall have the MADHeader:StatusField redirection-required bit set indicating that a ClassPortInfo attribute was returned rather than the expected attribute.

A response with the MADHeader:Status:RedirectionRequired bit set indicates that the request was not performed and that the request must be issued to the alternate interface specified in ClassPortInfo.

Redirection may be used at any time, so requesters should always be prepared to be redirected.

It is permissible for different requesters for the same management class on a channel adapter, switch, or router to be redirected to a different interface. The redirection operation is depicted in Figure 153 GSI Redirection on page 635.

Redirection information is also available by doing a normal Get() specifying the class of interest in the MADHeader:MgmtClass field of the MAD header and ClassPortInfo as the attribute.

The ClassPortInfo attribute contains all of the information necessary to access the redirected service either from within the same subnet or from a different subnet. ClassPortInfo may be programmed to include all of the parameters a source needs to form a complete GRH.

C13-44: A GRH shall be included in redirected class messages only if the RedirectGID component of ClassPortInfo is non zero.

It is the responsibility of any entity programming ClassPortInfo to assure that the parameters provided for accessing redirected services are consistent with address and access controls applicable to the redirected service.

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. The Status field is described in 13.4.7 Status Field on page 617.

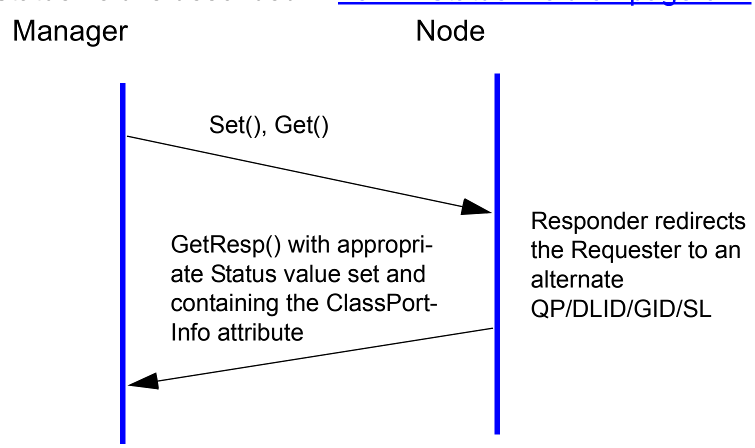


Figure 153 GSI Redirection

13.5.3 MAD VALIDATION

Packets arriving at a port of a channel adapter, switch, or router are validated according to the validation rules specified in 7.4 Data Packet Check on page 148 and 9.6 Packet Transport Header Validation on page 236. Only packets so validated are delivered to management entities. The contents of the data payload are further validated by management entities to validate that the data payload contains a valid MAD.

Valid MADS are delivered to appropriate management entities for processing.

13.5.3.1 MAD VALIDATION FOR SUBNET MANAGEMENT MADS

C13-45: Data payloads arriving at the SMI (QP 0) shall be validated as indicated in the bulleted list below. Packets failing one or more of these check are discarded and no action is taken in response unless the method is a Get() or a Set(). For Get() and Set() methods, the return of a GetResp() when validation has failed is optional.

- The data payload length must be 256 bytes
- LRH:VL must be 15
- BTH:QP must be 0
- BTH:OpCode must be Send only UD
- MADHeader:BaseVersion must be 1
- MADHeader:MgmtClass must specify a class of Subn or Directed Route Subn
- MADHeader:AttributeID must specify an attribute supported by the class specified in MADHeader:MgmtClass.

o13-18: If a GetResp() is returned, any conditions detected that have corresponding codes assigned in Table 102 MAD Common Status Field Bit Values on page 617 shall be reflected by corresponding settings of bits in MADHeader:StatusField in the response.

SMAs are not required to check the validity of the attribute content.

If a channel adapter, switch, or router supports a subnet manager, some MADs may be destined for the SMA while others may be destined for the SM. The discrimination between the SMA as a destination and the SM as a destination is based on the class, the method, and the attribute. See 14.2 Subnet Management Class on page 642. Table 108 SM MAD Sources and Destinations on page 636 indicates which SMPs originate at an SM, which SMPs originate at an SMA, and which SMPs may be destined to SMAs or to SMs.

Table 108 SM MAD Sources and Destinations

MAD Type	Source	Destination	Notes
Get(*)	SM	SMA	Applies for all attributes except SMInfo
Get(SMInfo)	SM	SM	Applies only for the SMInfo attribute
Set(*)	SM	SMA	Applies for all attributes except SMInfo
Set(SMInfo)	SM	SM	Applies only for the SMInfo attribute
GetResp(*)	SMA	SM	Applies for all attributes except SMInfo

Table 108 SM MAD Sources and Destinations

MAD Type	Source	Destination	Notes
GetResp(SMInfo)	SM	SM	Applies only for the SMInfo attribute
Trap()	SMA	SM	Applies to all subnet management traps.
TrapRepress()	SM	SMA	Applies to all subnet management traps.

This specification does not require that subnet managers be implemented in any particular way. It does require that subnet managers be able to originate and receive subnet management MADs. Implicitly, however an SM is realized, the mechanisms used must provide for the SM implementation to cause packets to be sent and received that have QP 0 as the source and destination QPs and which will be transmitted and received on VL 15.

C13-46: The SMI shall handle all Directed Route SMPs as described in 14.2.2 SMPs and Directed Route Algorithm on page 645.

13.5.3.2 MAD VALIDATION FOR SUBNET ADMINISTRATION AN GENERAL SERVICES

13.5.3.2.1 MAD VALIDATION AT THE GSI

C13-47: Data payloads arriving at the GSI (QP 1) shall be validated as specified in the bulleted list below. Packets failing one or more of these checks are discarded and no action is taken in response unless the method is a Get() or a Set(). For Get() and Set() methods, the return of a GetResp() when validation has failed is optional.

- The data payload length must be 256 bytes.
- LRH:VL must not be 15
- BTH:QP must be 1
- BTH:OpCode must be Send only UD
- MADHeader:Baseversion must be 1
- MADHeader:MgmtClass must specify a class supported on the channel adapter, switch, or router.

o13-19: If a GetResp() is returned, any conditions detected that have corresponding codes assigned in Table 102 MAD Common Status Field Bit Values on page 617 shall be reflected by corresponding settings of the bits in MADHeader:StatusField of the response.

It is not specified how GMPs passing through the GSI are dispatched to the appropriate class agents that are supported and which are not redirected.

If a class is supported and if redirection has been configured for that class, the response to a request arriving at the GSI containing MADHeader:MgmtClass of a redirected class is to reply with the redirection information for the class as specified in 13.5.2 GSI Redirection on page 634.

On CAs implementing the verbs layer specified in Chapter 11: Software Transport Verbs on page 473, GSMs may be implemented either below the verb layer or above the verb layer. For a GSM implemented above the verb layer and communicating via a source QP 1, it is not specified how disambiguation between GMPs destined to GSAs on that CA and GMPs destined to that GSM is performed. The basis for such differentiation is both class and context dependent as well as implementation dependent.

C13-48: If a CA does not support operation of a GSM via QP 1 from on top of its verb layer, that is, if it does not implement disambiguation of GMPs destined to a GSA below the verbs and GMPs destined to QP 1 associated with a GSM implemented above the verbs, it shall not permit QP 1 to be created above the verb layer.

See also 13.5.1.2 Processing General Services Management Packets (GMPs) on page 632.

Regardless of the implementation, the behavior of GSAs and GSMs with respect to the injection of messages on the wire, processing of messages from the wire, and responding to messages received must conform to the requirements of applicable sections in Chapter 16: General Services.

Implicitly, implementations of SAs, GSMs and GSAs must be able to send GMPs destined to QP 1.

13.5.3.2.2 MAD VALIDATION AT THE SA AND GSAS

Packets arriving at an SA or GSA via QP 1 have already been validated as properly formed MADs.

Packets arriving at a SA or GSA via any QP other than QP1 have been redirected. Such packets have not been validated as properly formed MADs.

o13-20: Agents processing GMPs that have been redirected shall first validate the GMPs as follows:

- The data payload length must be 256 bytes
- LRH:VL must not be 15
- The BTH:OpCode must be Send only UD
- MADHeader:BaseVersion must be 1
- MADHeader:MgmtClass must specify a class supported on the channel adapter, switch, or router.

C13-49: All packets arriving for processing at an SA or a GSA shall be further validated as follows:

- MADHeader:Method must specify a method valid for the specified class
- MADHeader:AttributeID must specify an attribute supported by the class specified in MADHeader:MgmtClass.

C13-50: GMPs failing one or more validity checks shall be discarded unless the method is one for which a response is normally returned (such as Get(), Set(), or Report()). The return of a response when validation has failed is optional.

o13-21: If a GetResp() is returned, any conditions detected that have corresponding codes assigned in Table 102 MAD Common Status Field Bit Values on page 617 shall be reflected by corresponding settings of the bits in the status field of the response.

GSAs are not required to check the validity of the attribute content.

Additional class specific checking requirements may be specified. Such requirements, if any, are defined in the class specific sections of Chapter 15: Subnet Administration on page 701 and Chapter 16: General Services on page 748.

13.5.4 RESPONSE GENERATION

Some methods require that the recipient return a response to the sender. This requires that the recipient be able to build a properly formed message which is consistent with the address and access rules associated with the sender.

In general, the sender may be in the same subnet or in a different subnet. Correspondingly, a request packet may or may not include a GRH.

C13-51: If the request packet does not contain a GRH, the response packet shall not contain a GRH and the response packet is constructed as follows:

- The SLID of the request packet shall be used as the DLID in the response packet.
- The Source QP of the request packet shall be used as the destination QP in the response packet.
- The SL specified in the request packet is used as the SL in the response packet.
- For GMPs, the responder's P_Key used to match the P_Key in the request packet is used as the P_Key in the response packet.

- Fields not otherwise specified in this section are filled in according to the requirements of the transport sections applicable to send only unreliable datagrams. See Chapter 9: Transport Layer on page 203.
- A GRH is not inserted in the response packet.

C13-52: If the original request packet contained a GRH, then the response packet must also contain a GRH. In this case the response packet is constructed as follows:

- The SGID in the GRH of the request packet shall be used as the DGID in the GRH of the response packet.
- FlowLabel and TrafficClass are copied without change from the GRH in the request packet to the GRH in the response packet.
- HopLimit in the GRH of the response packet is set to 0xFF.
- The SLID of the request packet shall be used as the DLID in the response packet.
- The Source QP of the request packet shall be used as the destination QP in the response packet.
- The SL specified in the request packet is used as the SL in the response packet.
- The responder's P_Key used to match the P_Key in the request packet is used as the P_Key in the response packet.
- Fields not otherwise specified in this section are filled in according to the requirements of the transport sections applicable to send only unreliable datagrams. See Chapter 9: Transport Layer on page 203.
- The GRH as formed above must be inserted in the response packet.

Note that GMP requests and responses, on the GSI or redirected, always use the well-known Q_Key (0x8001_0000) in the DETH.

Chapter 14: Subnet Management

14.1 Subnet Management Model

Each subnet has at least one subnet manager (SM). Each SM resides on a port of an CA, router, or switch and can be implemented either in hardware or software. When there are multiple SMs on a subnet, one SM will be the master SM. The remaining SMs must be standby SMs. There is only one SM per port.

The master SM is a key element in initializing and configuring an IB subnet. The master SM is elected as part of the initialization process for the subnet and is responsible for:

- Discovering the physical topology of the subnet
- Assigning Local Identifiers (LIDs) to the endnodes, switches, and routers
- Establishing possible paths among the endnodes
- Sweeping the subnet, discovering topology changes and managing changes as nodes are added and deleted.

The communication between the master SM and the SMAs, and among the SMs, is performed with subnet management packets (SMPs). SMPs provide a fundamental mechanism for subnet management.

There are two types of SMPs: LID routed and directed route. LID routed SMPs are forwarded through the subnet (by the switches) based on the LID of the destination. Directed route SMPs are forwarded based on a vector of port numbers that define a path through the subnet. Directed route SMPs are used to implement several management functions, in particular, before the LIDs are assigned to the nodes. SMPs are specified in section 14.2 Subnet Management Class on page 642.

Every switch, CA, and router has a subnet management agent (SMA), managed by the master SM. SMA are specified in section 14.3 Subnet Management Agent on page 682.

The details of operation for both master and standby SMs are described in section 14.4 Subnet Manager on page 687

14.2 SUBNET MANAGEMENT CLASS

This section defines the Subnet Management class of MADs. These MADs are also referred to as Subnet Management Packets, or SMPs. The purpose of this class is to provide subnet configuration, monitoring and query of nodes within a subnet. SMPs are exchanged between a SM and SMAs on the subnet as described in Table 108 SM MAD Sources and Destinations on page 636.

There are two management classes dedicated to subnet management.

C14-1: The subnet management classes shall be identified by the MAD-Header:MgmtClass value of 0x01 for the *LID Routed* class and 0x81 for the *Directed Route* class as listed in Table 100 Management Class Values on page 606.

This section will describe class-specific methods, attributes, standard header fields and protocols for the Subnet Management Classes.

14.2.1 DATAGRAM FORMATS AND USE

C14-2: The datagrams in this class shall conform to the MAD format and usage rules as specified in section 13.4.2 Management Datagram Format on page 604.

14.2.1.1 SMP DATA FORMAT - LID ROUTED

LID Routed SMPs are routed through the subnet using the normal switch forwarding tables set up during subnet initialization.

C14-3: A LID routed SMP shall have a format shown in Figure 154 on page 642 and Table 109 on page 643.

Figure 154 SMP Format (LID Routed)

bits / bytes	31-24	23-16	15-8	7-0
0	Common MAD Header			
...				
20				

Figure 154 SMP Format (LID Routed)

bits / bytes	31-24	23-16	15-8	7-0
24	M_Key			
28				
32	Reserved3 (32 bytes)			
...				
60				
64	SMP Data (64 bytes)			
128	Reserved4 (128 bytes)			
252				

Table 109 SMP Fields (LID Routed)

Object	Length	Description
Common MAD Header	24 bytes	Common MAD as described in 13.4.2 Management Datagram Format on page 604.
M_Key	8 bytes	A 64 bit key, which is employed for SM authentication. Usage is defined in section 14.2.4 Management Key on page 654.
Reserved3	32 bytes	For aligning the SMP data field with the directed route SMP data field. Set to all zeroes.
SMP Data	64 bytes	64 byte field of SMP data used to contain the method’s attribute.
Reserved4	128 bytes	Reserved. Shall be set to 0.

14.2.1.2 SMP DATA FORMAT - DIRECTED ROUTE

Directed route SMPs are routed through the subnet from SMA to SMA using a store-and-forward technique between neighboring nodes. They are therefore not dependent on routing table entries. Directed route SMPs are primarily used for discovering the physical connectivity of a subnet before it has been initialized.

C14-4: A Directed Routed SMP shall have a format shown in Figure 155 SMP Format (Directed Route) on page 644 and Table 110 SMP Fields (Directed Route) on page 644.

Figure 155 SMP Format (Directed Route)

bits / bytes	31-24	23-16	15-8	7-0
0	Common MAD Header1			
4	D | Status		Hop Pointer	Hop Count
...	Common MAD Header2			
20				
24	M_Key			
28				
32	DrSLID		DrDLID	
34	Reserved2 (28 bytes)			
...				
60				
64	SMP Data (64 bytes)			
128	Initial Path (64 bytes)			
192	Return Path (64 bytes)			
...				
252				

Table 110 SMP Fields (Directed Route)

Object	Length	Description
Common MAD Header1	4 bytes	Bytes 0-3 of the common MAD as described in 13.4.2 Management Datagram Format on page 604.
D	1 bit	Normally part of the class specific status field, this Direction bit is used by directed routing to determine direction of packet. If 0, the direction is outbound, from SM to node. If 1, the direction is inbound, from node to SM.
Status	15 bits	Code indicating status of method, as defined in 13.4.7 Status Field on page 617. There are no SMP status bits (bits 14-8 must be zero).
Hop Pointer	1 byte	Hop Pointer is used to indicate the current byte of the Initial/Return Path field.

Table 110 SMP Fields (Directed Route)

Hop Count	1 byte	Hop Count is used to contain the number of valid bytes in the Initial/Return Path. It indicates how many direct route 'hops' to take.
Common MAD Header2	16 bytes	Bytes 8-23 of common MAD as described in 13.4.2 Management Datagram Format on page 604.
M_Key	8 bytes	A 64-bit key, which is employed for SM authentication. Usage is defined in section 14.2.4 Management Key on page 654.
DrSLID	2 bytes	Directed route source LID. Used in directed routing.
DrDLID	2 bytes	Directed route destination LID. Used in directed routing.
Reserved2	28 bytes	For the purpose of aligning the Data field on a 64 byte boundary. Set to all all zeroes.
Data	64 bytes	64-byte field of SMP data used to contain the method's attribute.
Initial Path	64 bytes	64-byte field containing the initial directed path. Each byte in this field represents a port.
Return Path	64 bytes	64-byte field containing the returning directed path. Each byte in this field represents a port.

14.2.2 SMPs and Directed Route Algorithm

Directed route SMPs provide a mechanism for forwarding management packets throughout a configured, unconfigured, or partially configured subnet. This mechanism can be used to discover nodes in the subnet, perform diagnostics or verify link connectivity, bypassing the normal switch LID forwarding mechanism.

There are two components that support this mechanism in subnets: the Permissive destination address and directed routing. The Permissive destination address is defined in 4.1 Terminology And Concepts on page 116. When a node, including switches, receives a packet with this address it forwards it to its Subnet Management Interface. Directed routing permits the definition of an explicit route, based on intervening switch port numbers, that a packet is to transverse throughout the subnet.

Directed routing, in general, progresses much more slowly than normal switching. This is because each switch along the route has to perform some processing on every directed routed packet. Moreover, the IBA permits nodes to reserve a minimal amount of buffering for processing of SMPs. As a result, SMPs may be discarded in the subnet if the injection rate exceeds the buffering and processing capacity of the subnet and end-nodes. Therefore, it is recommended that directed routing only be used where necessary.

The directed routing algorithm provides a method to use normal LID routing on either side of the directed route. No support is provided for more than one directed route. It is not possible to specify two or more directed routes with intervening LID routes. This is illustrated in Figure 156 Complete route using directed routing on page 646. The complete route between two nodes is made up of three parts, each of them potentially empty:

- From the source node to the source switch. This part uses LID routing, the source node and the source switch are identified by their LIDs. There may be other switches between them but this portion of the subnet has already been configured to allow LID routing.
- From the source switch to the destination switch. This part uses direct routing. The route is specified by stating the port number a packet must use to leave a switch. This portion of the subnet need not have been configured to allow LID routing.
- From the destination switch to the destination node. This part uses LID routing, the destination node and the destination switch are identified by their LIDs. There may be other switches between them but this portion of the subnet has already been configured to allow LID routing.

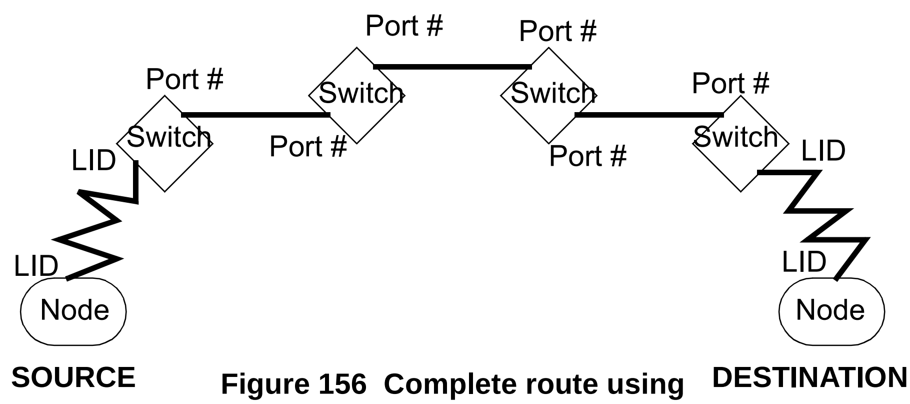


Figure 156 Complete route using directed routing

Since each part may be empty, there are eight combinations, although only four are really useful:

- All three parts are empty. This is used to loopback to oneself before a LID has been assigned. See Figure 157 Loopback using directed routing on page 647.

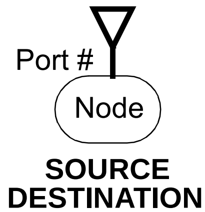


Figure 157 Loopback using directed routing

- Both LID routed parts are empty. This is a pure directed route used in a portion of a subnet not configured for LID routing. See Figure 158 Pure directed route on page 647.

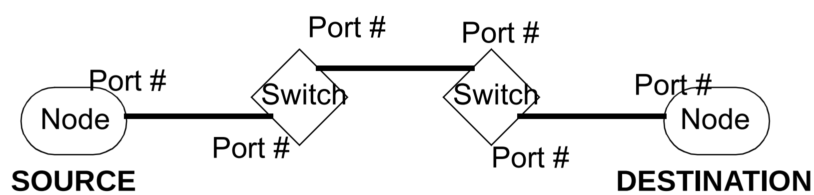


Figure 158 Pure directed route

- One of the LID routed part is empty. This is used when a portion of the subnet, either at the source or at the destination, has been configured for LID routing. See Figure 159 Directed route with LID routing part at the source on page 648 and Figure 160 Directed route with

LID routing part at the destination on page 648.

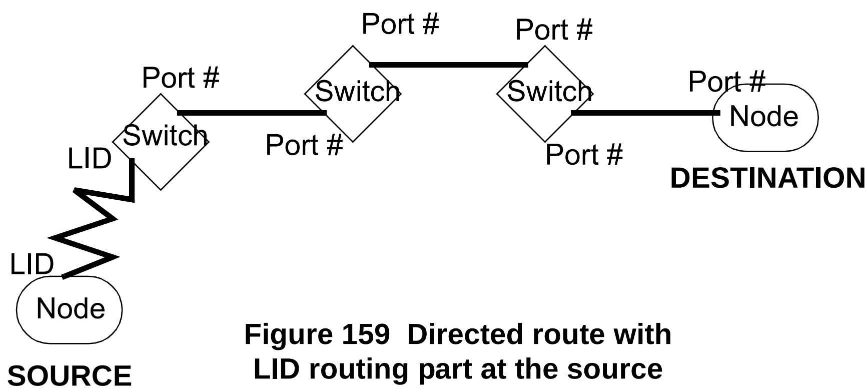


Figure 159 Directed route with LID routing part at the source

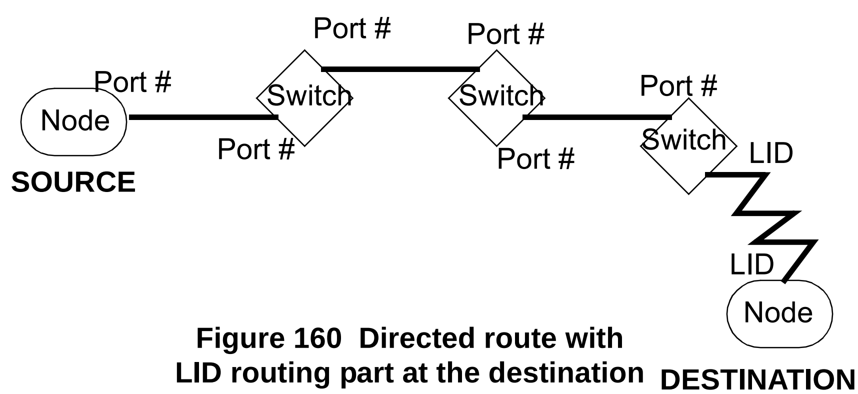


Figure 160 Directed route with LID routing part at the destination

- No part is empty. This is the general case as illustrated in Figure 156 Complete route using directed routing on page 646 when portions of the subnet have been initialized both at the source and destination but not in between.

The following section describes how a directed route packet is initialized, how a return packet is initialized, and the algorithm used along the route by switches to forward the packets. Note that in the switched portions of a route, the nodes are not directly involved - the packet is switched along the path just as any other packet is.

14.2.2.1 OUTGOING DIRECTED ROUTE SMP INITIALIZATION

C14-5: Only a SM shall originate a directed route SMP.

C14-5.a1: The SM shall insure that all nodes on a LID routed segment shall be properly initialized for LID routing.

In the following sections, the source node where the SM originator resides is called the requestor node, and the destination node is called the responder node, even when describing the return process. When directed route is used, it refers to the directed route part only, not the complete route.

C14-6: The fields of the directed route SMP shall be initialized as follows:

1) Mgmt Class shall be set to the directed route Subnet Management class as specified in Table 100 Management Class Values on page 606.
2) Method shall be set to *SubnGet()* or *SubnSet()* as specified in Table 101 Common Management Methods on page 608.
3) D bit shall be set to 0.
4) Hop Pointer shall be set to 0.
5) Hop Count shall be set to the number of hops, i.e. inter-switch links, along the directed route part. Valid values are from 0 to 63. In Figure 156 Complete route using directed routing on page 646, this number would be 3. In Figure 157 Loopback using directed routing on page 647, this number would be 0.
6) If the directed route part starts from the requestor node, i.e. there is no LID routed part at the source as illustrated in Figure 157 Loopback using directed routing on page 647, Figure 158 Pure directed route on page 647 or Figure 160 Directed route with LID routing part at the destination on page 648, then the DrSLID shall be set to the Permissive LID. If the directed route does *not* start from the requestor node, then DrSLID shall be set to the LID of the requestor node, which must have been assigned.
7) If the directed route ends at the responder node, i.e. there is no LID routed part at the destination as illustrated in Figure 157 Loopback using directed routing on page 647, Figure 158 Pure directed route on page 647 or Figure 159 Directed route with LID routing part at the source on page 648, then the DrDLID shall be set to the Permissive LID. If the directed route does *not* end at the responder node, then the DrDLID shall be set to the LID of the responder node, which must have been assigned.
8) Initial Path shall be set to an array of Hop Count port numbers corresponding to the ports at the starting end of hops, specifically, the port

from which the SMP will start travelling on the inter-node link along the directed route. The array shall be laid out in the initial Path field so that the byte at offset 0 in that field is reserved and the following bytes are filled with the port numbers in order.

9) Return Path shall be set to an array of Hop Count zeroes. The array shall be laid out in the Return Path field so that the byte at offset 0 in that field is reserved and the following bytes are filled with zeroes.

10) All other fields shall be set the same way they are for a LID routed SMP.

C14-7: The data packet headers for the unreliable datagram encapsulating the directed route SMP shall be initialized as follows:

1) If the directed route part starts from the requestor node, the SLID shall be set to the Permissive LID or a LID of this port. If the directed route does *not* start from the requestor node, the SLID shall be set to the LID of the requestor node, which must have been assigned.

2) DLID shall be set to the Permissive LID if the directed route part starts from the requestor node. If not, it shall be set to the LID of the source switch in the directed route part. That LID must have been assigned and routing must have been initialized between that switch and the requestor node.

3) All other fields shall be set the same way they are for a LID routed SMP.

The SM will then hand the packet to the SMI. If the DLID is the Permissive LID, the SMI processes the packet as described in section 14.2.2.2 Outgoing Directed Route SMP handling by SMI on page 650. If the DLID is *not* the Permissive LID, the SMI will output the packet as it does any LID routed packet.

14.2.2.2 OUTGOING DIRECTED ROUTE SMP HANDLING BY SMI

C14-8: Any SMP arriving at the SMI with a *MADHeader:MgmtClass* set to 0x81 (Directed Route class) shall be processed by the SMI.

C14-9: The SMI shall handle outgoing directed route SMPs (D bit is 0) as defined by the following steps:

1) If HopCount is non zero and Hop Pointer is less than Hop Count (in the range between 0 to Hop Count -1):

   The SMI shall alter the contents of the directed route SMP, in order, as follows:

a) If Hop Pointer is more than 0, if this node is not a switch, the SMI shall discard the SMP. If the Hop Pointer is more than 0 and the node is a switch, the entry indexed by Hop Pointer in the Return Path array of port numbers shall be set to the port number where the SMP was received.

b) The Hop Pointer shall be incremented by 1.

c) All other fields shall remain unchanged.

The data packet headers for the unreliable datagram encapsulating the directed route SMP shall be altered as follows:

d) For switches, the SLID shall be set to the Permissive LID. For CA's the SLID shall be set to a LID of this port or the Permissive LID.

e) The DLID shall be set to the Permissive LID.

The SMI shall output the packet on the port whose number is in the entry indexed by Hop Pointer in the Initial Path. If that port number is invalid, the SMI shall discard the SMP.

2) If Hop Pointer is equal to Hop Count, the SMI is the last one on the directed route part. The SMI shall alter the contents of the directed route SMP as follows:

a) The entry indexed by Hop Pointer in the Return Path array of port numbers shall be set to the port number where the SMP was received if Hop Pointer is non-zero.

b) The Hop Pointer shall be incremented by 1.

c) All other fields shall remain unchanged.

The SMI shall alter the data packet headers for the unreliable datagram encapsulating the directed route SMP in order, as follows:

d) For switches, the SLID of the outgoing directed route SMP shall be altered as follows: If the DrDLID is the Permissive LID, the SLID shall be set to the Permissive LID. If the DrDLID is *not* the Permissive LID, the SLID shall be set to the LID of this node. For CA's, if the DrDLID is the Permissive LID, the SLID may be set to the Permissive LID; otherwise, the SLID shall be set to a LID of the port.

e) DLID shall be set to the DrDLID.

If the DLID is the Permissive LID, this node is the responder node and the SMI shall hand the packet to the SMA or SM, which may check that Hop Pointer is equal to Hop Count+1. If the DLID is *not* the Permissive LID, the SMI will output the packet as it does any LID routed packet if this node is a switch (if this node is *not* a switch, the SMP shall be silently discarded).

3) If Hop Pointer is equal to Hop Count+1, this node is the responder node and the SMI shall hand the packet to the SMA or SM, which may check that Hop Pointer is equal to Hop Count+1.

4) If Hop Pointer is greater than Hop Count+1 (Hop Count+2 to 255), the SMI shall silently discarded the SMP.

The handling of returning directed route SMPs (D bit is 1) is described in section 14.2.2.3 Returning Directed Route SMP Initialization on page 652.

14.2.2.3 RETURNING DIRECTED ROUTE SMP INITIALIZATION

The SMA or SM receiving a directed route SMP processes it (with regard to handling of the method and attribute) as it does a LID routed SMP. The receiving SMA or SM may determine that it should send a response.

C14-10: The fields of the directed route response SMP shall be initialized as follows:

1) Method shall be set to SubnGetResp() as specified in Table 101 Common Management Methods on page 608.

2) D bit shall be set to 1.

3) Mgmt Class, Hop Pointer, Hop Count, DrSLID, DrDLID, Initial Path and Return Path shall be copied as is from the request SMP.

4) All other fields shall be set the way they are set for a LID routed SMP.

C14-11: The data packet headers for the unreliable datagram encapsulating the directed route response SMP shall also be initialized as follows:

1) If the directed route part starts from the responder node, the SLID shall be set to the Permissive LID or a LID of this port. If the directed route does not start from the responder node, the SLID shall be set to the LID of the responder node, which must have been assigned.

2) The DLID shall be set to the SLID of the directed route request SMP.

3) All other fields shall be set the way they are set for a LID routed SMP.

The SMA or SM will then hand the packet to the SMI. If the DLID is not the Permissive LID, the SMI will output the packet as it does any LID routed packet. If the DLID is the Permissive LID, the SMI processes the packet as described in the following section.

14.2.2.4 RETURNING DIRECTED ROUTE SMP HANDLING BY SMI

C14-12: This compliance statement is obsolete and has been removed.

C14-13: The SMI shall handle returning directed route SMPs (D bit is 1) as defined by the following steps:

1) If HopCount is non-zero and Hop Pointer is more than 1 (2 to Hop Count+1):

   The SMI shall alter the contents of the directed route SMP, in order, as follows:

   a) If Hop Pointer is less than Hop Count + 1, and if this node is not a switch, the SMI shall discard the SMP.

   b) Hop Pointer shall be decremented by 1.

   c) All other fields shall remain unchanged.

   The SMI shall alter the data packet headers for the unreliable datagram encapsulating the directed route SMP as follows:

   d) For switches, the SLID shall be set to the Permissive LID. For CA's the SLID shall be set to a LID of this port or the Permissive LID.

   e) DLID shall be set to the Permissive LID.

   The SMI shall output the packet on the port whose number is in the entry indexed by Hop Pointer in the Return Path. If that port number is invalid, the SMI shall discard the SMP

2) If Hop Pointer is equal to 1, the SMI is the last one on the directed route part. The SMI shall alter the fields of the directed route SMP, in order, as follows:

   a) Hop Pointer shall be decremented by 1.

   b) All other fields shall remain unchanged.

   The SMI shall alter data packet headers for the unreliable datagram encapsulating the directed route SMP, in order, as follows:

   c) For switches, the SLID of the returning directed route SMP shall be altered as follows: If the DrSLID is the Permissive LID, the SLID shall be set to the Permissive LID. If the DrSLID is not the Permissive LID, the SLID shall be set to the LID of this node. For CA's, if the DrSLID is the Permissive LID, the SLID may be set to the Permissive LID; otherwise, the SLID shall be set to a LID of the port.

   d) DLID shall be set to the DrSLID.

   If the DLID is the Permissive LID, then this node is the requester node and the SMI must hand the packet to the SM which may check that Hop Pointer is equal to 0.

   If the DLID is *not* the Permissive LID and if this node is a switch, the SMI will output the packet as it does any LID routed packet. If the DLID is not the Permissive LID and this node is *not* a switch, then the SMP shall be silently dropped.

3) If Hop Pointer is equal to 0, this node is the requestor node and the SMI must hand the packet to the SM, which may check that Hop Pointer is equal to 0.

4) If Hop Pointer is in the range (HopCount+2) to 255, then the SMI shall silently discard the SMP.

The handling of outgoing directed route SMPs (D bit is 0) is described in section 14.2.2.2 Outgoing Directed Route SMP handling by SMI on page 650.

14.2.3 METHODS

The Subnet Management class uses a subset of the common methods described in section 13.4.5 Management Class Methods on page 607.

Table 111 Subnet Management Methods

Method Type	Value	Description
SubnGet()	0x01	Request a get (read) of an attribute.
SubnSet()	0x02	Request a set (write) of an attribute.
SubnGetResp()	0x81	Response from a get or set request.
SubnTrap()	0x05	Notify an event occurred.
SubnTrapRepress()	0x07	Cease sending repeated Trap.

Table 108 SM MAD Sources and Destinations on page 636 indicates which methods are applied to SMPs that originate at a SM, SMPs that originate at a SMA, and SMPs that may be destined to SMAs or to SMs.

C14-14: This compliance statement is obsolete and has been removed.

Subnet Management entities, the SMA and SM, support the methods listed in Table 111 Subnet Management Methods on page 654.

14.2.4 MANAGEMENT KEY

SMPs are used to initialize and configure CAs, switches and routers, and are therefore considered privileged operations. As a result, there is a mechanism provided to authorize subnet management operations based on:

- a Key stored in the *MADHeader:M_Key* of the LID routed and Directed route subnet management class datagram as shown in Figure 154 SMP Format (LID Routed) on page 642 and Figure 155 SMP Format (Directed Route) on page 644, respectively.
- a Key kept locally on each port in the *PortInfo:M_Key* component of the *PortInfo* attribute that is described in Table 126 PortInfo on page 665.

Authentication is performed by the management entity at the destination port and is achieved by comparing the key contained in the SMP with the key residing at the destination port. This key is known as the Management Key (M_Key).

C14-15: A M_Key contained in the *MADHeader:M_Key* of the SMP shall not be checked at the receiving port with the *PortInfo:M_Key* set to zero. As a result, no authentication is performed.

If the *PortInfo:M_Key* is nonzero, authentication at the receiving port and access to the port attributes is determined by the contents of the *PortInfo:M_KeyProtectBits* as described in section 14.2.4.1 Levels of Protection on page 655. Finally, M_Keys can be lost, so Key recovery is provided by the *PortInfo:M_KeyLeasePeriod* components and is described in section 14.2.4.2 Lease Period on page 656.

14.2.4.1 LEVELS OF PROTECTION

C14-16: If the *PortInfo:M_Key* is non-zero, the management entity residing at the port shall perform authentication determined by the contents of the *PortInfo:M_KeyProtectBits* and the behaviors described in Table 112 Protection Levels on page 655.

Table 112 Protection Levels

PortInfo:M_KeyProtectBits	Description
0	*SubnGet(*)* shall succeed for any key in the *MADHeader:M_Key* and *SubnGetResp(PortInfo)* shall return the contents of the *PortInfo:M_Key* component. *SubnSet(*)* and *SubnTrapRepress(*)* shall fail if *MADHeader:M_Key* does not match the *PortInfo:M_Key* component in the port.
1	*SubnGet(*)* shall succeed for any key in the *MADHeader:M_Key* and *SubnGetResp(PortInfo)* shall return the contents of the *PortInfo:M_Key* component set to zero if *MADHeader:M_Key* does not match the *PortInfo:M_Key* component in the port. *SubnSet(*)* and *TrapRepress(*)* shall fail if *MADHeader:M_Key* does not match the *PortInfo:M_Key* component in the port.

Table 112 Protection Levels

PortInfo:M_KeyProtectBits	Description
2	*SubnGet(*)*, *SubnSet(*),* and *TrapRepress(*)* shall fail if *MAD-Header:M_Key* does not match the *PortInfo:M_Key* component in the port.
3	*SubnGet(*)*, *SubnSet(*),* and *TrapRepress(*)* shall fail if *MAD-Header:M_Key* does not match the *PortInfo:M_Key* component in the port.

14.2.4.2 LEASE PERIOD

A Lease Period is specified by setting the contents of the *Port-Info:M_KeyLeasePeriod* component. It is intended to allow an M_Key to 'expire' if the master SM inadvertently goes away without sharing the M_Key with backup SMs and there is no other out-of-band recovery mechanism available.

C14-17: The lease period timer shall start counting down toward zero on a port when a SMP is received for which the M_Key check was performed according to Table 112 Protection Levels on page 655 and failed. If the lease timer countdown is already underway, it shall not be interrupted by the arrival of that SMP.

C14-18: The PortInfo:M_KeyViolations component shall be incremented on a port when a SMP is received for which the M_Key check was performed according to Table 112 and failed. The incrementing shall stop when the component reaches all 1s.

Furthermore, if the port is capable of sending traps, a M_Key violation trap described in Table 115 Traps on page 660 may be sent to the master SM indicating that the lease timer has started counting down. In response to that trap, the master SM may refresh the Lease Period. If the master SM that originally set the M_Key has gone away, the Lease Period may expire.

C14-19: The lease period counter shall cease counting down and shall be reset to the value contained in *PortInfo:M_KeyLeasePeriod* component on a port when any SMP is received with *MADHeader:M_Key* that matches the *PortInfo:M_Key.*

C14-20: The *PortInfo:M_KeyProtectBits* shall be set to zero when the lease period counter transitions from non-zero to zero.

When the lease period expires, clearing the M_Key Protection bits will allow any SM to read (and then set) the M_Key.

C14-21: When the *PortInfo:M_KeyLeasePeriod* is set to zero, the lease period shall never expire.

Whether there is an out-of-band mechanism to reset data protected with a lease period of zero is outside the scope of the specification.

14.2.4.3 NOTES ON EXPECTED USAGE

- The SM is responsible for keeping track of the M_Keys for the nodes that it is managing, to make sure that it uses the correct key for each node.
- If standby SMs exist in the subnet for redundancy, then the M_Keys may be shared so that failover to another SM can be accommodated easily.
- An SM may have exclusive access to a node (or set of nodes), by using an M_Key which is only known by that SM and the particular node(s).
- *SubnSet()* is always protected by this mechanism as it can affect the state of the node. *SubnGet()* is protected only if *PortInfo:M_KeyProtectBits* is appropriately set.

14.2.4.4 UPDATE PROCEDURE

Node protection/ownership is assigned in one “atomic” operation.

C14-22: The *PortInfo:M_Key*, the *PortInfo:M_KeyProtectBits*, the *PortInfo:M_KeyLeasePeriod* components in the PortInfo Attribute shall be set in one *SubnSet(PortInfo)* method.

A returned *SubnGetResp(PortInfo)* with a status of zero indicates to the SM that it has taken ownership of the node.

14.2.4.5 INITIALIZATION

C14-23: When initially powered-up or reset, the *PortInfo:M_Key*, the *PortInfo:M_KeyProtectBits*, the *PortInfo:M_KeyLeasePeriod* components of an endport shall be set to zero if NVRAM is not used or to a value stored in NVRAM.

If the M_Key related components are not stored in NVRAM, the *PortInfo:M_Key*, the *PortInfo:M_KeyProtectBits*, the *PortInfo:M_KeyLeasePeriod* components may be set by any master SM during subnet initialization. Initialization of M_KeyLeasePeriod to a value of zero (infinite) notwithstanding, whenever a port's M_Key-related components are not stored in NVRAM, any subnet manager can successfully read and then set the port’s M_Key during subnet initialization.

14.2.4.6 SMI

The SMI will not check the M_Key in the header of a SMP since that is the responsibility of the management entities that reside behind the SMI.

14.2.5 ATTRIBUTES

In the SMP, attributes can be up to 64 bytes long. The Table 113 Subnet Management Attributes (Summary) on page 658 summarizes the subnet management attributes and Table 114 Subnet Management Attribute / Method Map on page 659 indicates which methods apply to each attribute.

C14-24: Subnet management entities shall support the attributes and methods as listed in Table 113 Subnet Management Attributes (Summary) on page 658 and Table 114 Subnet Management Attribute / Method Map on page 659.

Table 113 Subnet Management Attributes (Summary)

Attribute Name	Attribute ID	Attribute Modifier	Description	Required For
Notice	0x0002	0x0000_0000	Information regarding the associated Notice or Trap	Optional[a]
NodeDescription	0x0010	0x0000_0000	Node Description String	All Nodes
NodeInfo	0x0011	0x0000_0000	Generic Node Data	All Nodes
SwitchInfo	0x0012	0x0000_0000	Switch Information	Switches
GUIDInfo	0x0014	GUID Block	Assigned GUIDs	All Endports
PortInfo	0x0015	Port Number	Port Information	All Ports on All Nodes
PartitionTable	0x0016	Port Number/P_Key block	Partition Table	All Ports on All Nodes
SLtoVLMappingTable	0x0017	Input/Output Port Number	Service Level to Virtual Lane mapping Information	All Ports on All Nodes (optional[b])
VLArbitration	0x0018	Output Port/Component	List of Weights	All Ports on All Nodes (optional[c])
LinearForwardingTable	0x0019	LID Block	Linear Forwarding Table Information	Switches (optional[d])
RandomForwardingTable	0x001A	LID Block	Random Forwarding Database Information	Switches (optional[d])
MulticastForwardingTable	0x001B	LID Block	Multicast Forwarding Database Information	Switches (optional)

Table 113 Subnet Management Attributes (Summary)

Attribute Name	Attribute ID	Attribute Modifier	Description	Required For
SMInfo	0x0020	0x0000_0000 - 0x0000_0005	Subnet Management Information	All nodes hosting an SM
VendorDiag	0x0030	0x0000_0000 - 0x0000_FFFF	Vendor Specific Diagnostic	All Ports on All Nodes
LedInfo	0x0031	0x0000_0000	Turn on/off LED	All nodes
	0xFF00-0xFFFF	0x0000_0000 - 0x0000_FFFF	Range reserved for Vendor Specific attributes.	

a. Either Notices or Traps or both are required.
b. Optional on ports that support only one data VL.
c. Prohibited on ports that support only one data VL.
d. LinearForwardingTable and RandomForwardingTable are mutually exclusive, but one is required.

Table 114 Subnet Management Attribute / Method Map

Attribute Name	Get	Set	Trap
Notice	x	x	x
NodeDescription	x		
NodeInfo	x		
SwitchInfo	x	x	
GUIDInfo	x	x	
PortInfo	x	x	
PartitionTable	x	x	
SLtoVLMappingTable	x	x	
VLArbitration	x	x	
LinearForwardingTable	x	x	
RandomForwardingTable	x	x	
MulticastForwardingTable	x	x	
SMInfo	x	x	
VendorDiag	x		
LedInfo	x	x	

14.2.5.1 NOTICES AND TRAPS

This attribute is a common attribute described in section 13.4.8.2 Notice on page 622. The following traps are defined for the Subnet Management class.

Table 115 Traps

Trap Number	Sending Node Type	DataDetails
64	subnet	<LIDADDR><PORTNO> is now in service
65	subnet	<LIDADDR><PORTNO> is out of service
128	switch	*Link state of at least one port of switch at* <LIDADDR> *has changed.*
129	any	Local Link Integrity threshold reached at <LIDADDR><PORTNO>
130	any	Exccessive Buffer Overrun threshold reached at <LIDADDR><PORTNO>
131	switch	Flow Control Update watchdog timer expired at <LIDADDR><PORTNO>
256	any	*Bad M_Key,* <MKEY> *from* <LIDADDR> *attempted* <METHOD> with <ATTRIBUTEID> and <ATTRIBUTEMODIFIER>.
257	any	Bad P_Key, <KEY> from <LIDADDR1> /<GIDADDR1>/<QP1> to <LIDADDR2>/<GIDADDR2>/<QP2> on <SL>.
258	any	Bad Q_Key, <KEY> from <LIDADDR1>/<GIDADDR1>/<QP1> to <LIDADDR2>/<GIDADDR2>/<QP2> on <SL>.

Traps use the following layout for the DataDetails component of the Notice attribute. Fields shall be filled with the information corresponding to the description of a given trap.

Table 116 Notice DataDetails For Traps 64 and 65

Field	Length(bits)	Description
LIDADDR	16	Local Identifier
PORTNO	8	Port number
Padding	408	Shall be ignored on read. Content is unspecified.

Table 117 Notice DataDetails For Trap 128

Field	Length(bits)	Description
LIDADDR	16	Local Identifier
Padding	416	Shall be ignored on read. Content is unspecified.

Table 118 Notice DataDetails For Traps 129, 130 and 131

Field	Length(bits)	Description
Reserved0	16	Shall be filled with zeroes
LIDADDR	16	Local Identifier
PORTNO	8	Port number
Padding	392	Shall be ignored on read. Content is unspecified.

Table 119 Notice DataDetails For Trap 256

Field	Length(bits)	Description
Reserved0	16	Shall be filled with zeroes
LIDADDR	16	Local Identifier
Reserved1	16	Shall be filled with zeroes
METHOD	8	Method
Reserved2	8	Shall be filled with zeroes
ATTRIBUTEID	16	Attribute ID
ATTRIBUTEMODIFIER	32	Attribute Modifier
MKEY	64	M_Key
Padding	256	Shall be ignored on read. Content is unspecified.

Table 120 Notice DataDetails For Traps 257 and 258

Field	Length(bits)	Description
Reserved0	16	Shall be filled with zeroes
LIDADDR1	16	Local Identifier
LIDADDR2	16	Local Identifier
KEY	32	Q_Key or P_Key. If P_Key, the 16 most significant bits of the field shall be set to 0 and the 16 least significant bits of the field shall be set to the P_Key.
SL	4	Service Level
Reserved2	4	Must be filled with zeroes
QP1	24	Queue Pair
Reserved3	8	Must be filled with zeroes

Table 120 Notice DataDetails For Traps 257 and 258

Field	Length(bits)	Description
QP2	24	Queue Pair
GIDADDR1	128	Global Identifier. If no GRH is present in the offending packet, this field shall be filled with zeroes.
GIDADDR2	128	Global Identifier. If no GRH is present in the offending packet, this field shall be filled with zeroes.
Padding	32	Shall be ignored on read. Content is unspecified.

14.2.5.2 NodeDescription

Table 121 NodeDescription

Component	Access	Length(bits)	Description
NodeString	RO	512	UTF-8 encoded string to describe node in text format.

The contents of the NodeDescription attribute are the same for all ports on a nodes.

14.2.5.3 NodeInfo

The NodeInfo Attribute provides fundamental management information common to all CAs, routers, and switches. It shall be implemented by all nodes.The value of some NodeInfo components varies by port within a node.

Table 122 NodeInfo

Component	Access	Length (bits)	Offset (bits)	Description
BaseVersion[a]	RO	8	0	Supported MAD Base Version. Indicates that this node supports up to and including this version. Set to 1.
ClassVersion[a]	RO	8	8	Supported Subnet Management Class (SMP) Version. Indicates that this node supports up to and including this version. Set to 1.
NodeType[a]	RO	8	16	1: Channel Adapter 2: Switch 3: Router 0, 4 - 255: Reserved
NumPorts[a]	RO	8	24	Number of physical ports on this node.

Table 122 NodeInfo

Component	Access	Length (bits)	Offset (bits)	Description
Reserved	RO	64	32	Reserved, shall be zero.
NodeGUID[a]	RO	64	96	GUID of the HCA, TCA, switch, or router itself. All ports on the same node shall report the same Node-GUID. Provides a means to uniquely identify a node within a subnet and determine co-location of ports.
PortGUID[b]	RO	64	160	GUID of this port itself. One port within a node can return the NodeGUID as its PortGUID if the port is an integral part of the node and is not field-replaceable.
PartitionCap[a]	RO	16	224	Number of entries in the Partition Table for CA, router, and the switch management port. This is at a minimum set to 1 for all nodes including switches.
DeviceID[a]	RO	16	240	Device ID information as assigned by device manufac-turer.
Revision[a]	RO	32	256	Device revision, assigned by manufacturer.
LocalPortNum	RO	8	288	The number of the link port which received this SMP.
VendorID[a]	RO	24	296	Device vendor, per IEEE.

a. Value shall be the same for all ports on a node.
b. Value shall differ for each end port on a CA or router, but the same for all ports of a switch.

14.2.5.4 SwitchInfo

The SwitchInfo Attribute provides management information specific to switch nodes. It shall be implemented by all switches.

Table 123 SwitchInfo

Component	Access	Length (bits)	Offset (bits)	Description
LinearFDBCap	RO	16	0	Number of entries supported in the Linear Unicast Forwarding Table (starting at LID=0x0000 going up). LinearFDBCap = 0 indicates that there is no Linear Forwarding Database.
RandomFDBCap	RO	16	16	Number of entries supported in the Random Unicast Forwarding Table. RandomFDBCap = 0 indicates that there is no Random Forwarding Database.
MulticastFDBCap	RO	16	32	Number of entries supported in the Multicast Forwarding Table (starting at LID=0xC000 going up).
LinearFDBTop	RW	16	48	Indicates the top of the linear forwarding table. Packets received with unicast DLIDs greater than this value are discarded by the switch. This component applies only to switches that implement linear forwarding tables and is ignored by switches that implement random forwarding tables.

Table 123 SwitchInfo

Component	Access	Length (bits)	Offset (bits)	Description
DefaultPort	RW	8	64	Forward to this port all the unicast packets from the other ports whose DLID does not exist in the random forwarding table, see section Chapter 18:: Switches
DefaultMulticastPrimaryPort	RW	8	72	Forward to this port all the multicast packets from the other ports whose DLID does not exist in the forwarding table, see section 18.2.4.3.3 Required Multicast Relay on page 856.
DefaultMulticast-NotPrimaryPort	RW	8	80	Forward to this port all the multicast packets from the Default Primary port whose DLID does not exist in the forwarding table, see section 18.2.4.3.3 Required Multicast Relay on page 856.
LifeTimeValue	RW	5	88	Sets the time a packet can live in the switch, see section 18.2.5.4 Transmitter Queueing on page 860.
PortStateChange	RW	1	93	It is set to one anytime the *PortState* component in the PortInfo of any ports transitions from Down to Initialize, Initialize to Down, Armed to Down, or Active to Down as a result of link state machine logic. Changes in Portstate resulting from SubnSet do no change this bit. This bit is cleared by writing one, writing zero is ignored.
Reserved	RO	2	94	Reserved, shall be zero.
LIDsPerPort	RO	16	96	Specifies the number of LID/LMC combinations that may be assigned to a given external port for switches that support the Random Forwarding table.
PartitionEnforcementCap	RO	16	112	Specifies the number of entries in the partition enforcement table per physical port. Zero indicates that partition enforcement is not supported by the switch.
InboundEnforcementCap	RO	1	128	Indicates switch is capable of partition enforcement on received packets
OutboundEnforcementCap	RO	1	129	Indicates switch is capable of partition enforcement on transmitted packets
FilterRawPacketInboundCap	RO	1	130	Indicates switch is capable of raw packet enforcement on received packets
FilterRawPacketOutboundCap	RO	1	131	Indicates switch is capable of raw enforcement on transmitted packets

14.2.5.5 GUIDINFO

The GUIDInfo Attribute provides the means for setting the assigned local scope EUI-64 identifiers of channel adapters, routers, and switch management ports. These local scope EUI-64 identifiers are concatenated with a subnet prefix to form GIDs that are described in section 4.1.1 GID Usage and Properties on page 117.

The Attribute Modifier is a pointer to a block of 8 GUIDs to which this attribute applies. Valid values are from 0 to 31 and are further limited by the size of the *GUIDCap* of the port. Any entries in the block beyond the end of the GUID table are ignored on write and read back as zero. The block element at offset zero is read-only and is a copy of the *PortGUID* component.

The attribute selected corresponds to the port that received the SMP.

Table 124 GUIDInfo

Component	Access	Length(bits)	Description
GUIDBlock	RW	512	List of 8 GUID Block Elements.

Table 125 GUID Block Element

Component	Length(bits)	Description
GUID	64	GUID to be assigned to port.

14.2.5.6 PORTINFO

The PortInfo Attribute provides port-specific management information. It shall be implemented for every port on a node. Note that the value of some NodeInfo components varies by node type and by port within a node.

The Attribute Modifier selects the port that the operation specified by the SMP is performed. For switches, channel adapters, and routers, the range of values between 0 to N, where N is the number of ports and:

- For channel adapters and routers the value of zero indicates that the operation is performed on the port that received the SMP. Otherwise, if the value is non-zero and does not match the port number where the SMP is received, the PortInfo attribute is RO and the M_Key is checked for both the port where the SMP was received and the port selected by the attribute modifier.
- For switches, a value of zero selects the management port. Otherwise, if the value is non-zero, a physical port is selected.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
M_Key[a]	RW	64	0	The 8-byte management key. See section 14.2.4 Management Key on page 654.
GidPrefix[a]	RW	64	64	GID prefix for this port.
LID[a]	RW	16	128	The base LID of this port.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
MasterSMLID[a]	RW	16	144	The LID of the master SM that is managing this port.
CapabilityMask[a]	RO	32	160	Supported capabilities of this node. A bit set to 1 for affirmation of supported capability. 0: Reserved, shall be zero 1: IsSM 2: IsNoticeSupported 3: IsTrapSupported 4: IsResetSupported 5: IsAutomaticMigrationSupported 6: IsSLMappingSupported 7: IsMKeyNVRAM (supports M_Key in NVRAM) 8: IsPKeyNVRAM (supports P_Key in NVRAM) 9: IsLEDInfoSupported 10: IsSMdisabled 11 - 15: Reserved, shall be zero 16: IsConnectionManagementSupported 17: IsSNMPTunnelingSupported 18: Reserved, shall be zero 19: IsDeviceManagementSupported 20: IsVendorClassSupported 21 - 31: Reserved, shall be zero
DiagCode[a]	RO	16	192	Diagnostic code, as described in section 14.2.5.6.1 on page 671.
M_KeyLeasePeriod[a]	RW	16	208	Specifies the initial value of the lease period timer in seconds. The lease period is the length of time that the M_Key Protection bits are to remain non zero after a SubnSet(PortInfo) fails a M_Key check.See section 14.2.4 Management Key on page 654.
LocalPortNum	RO	8	224	The number of the link port which received this SMP.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
LinkWidthEnabled[b]	RW	8	232	Enabled link width, indicated as follows: 0: No State Change (NOP) 1: 1x 2: 4x 3: 1x or 4x 8: 12x 9: 1x or 12x 10: 4x or 12x 11: 1x, 4x or 12x 4 - 7, 12 - 254: Reserved (Ignored) 255: Set to LinkWidthSupported value. When writing this field, only legal transitions are valid. See Volume 2.
LinkWidthSupported[b]	RO	8	240	Supported link width, indicated as follows: 1: 1x 3: 1x or 4x 11: 1x, 4x or 12x 0, 2, 4-10, 12-255: Reserved
LinkWidthActive[b]	RO	8	248	Currently active link width, indicated as follows: 1: 1x 2: 4x 8: 12x 0, 3, 4-7, 9-255: Reserved
LinkSpeedSupported[b]	RO	4	256	Supported link speed, indicated as follows: 1: 2.5Gbps 0, 2 - 15: reserved
PortState[b]	RW	4	260	Port State. Enumerated as: 0: No State Change (NOP) 1: Down (includes failed links) 2: Initialize 3: Armed 4: Active 5: 15: Reserved - ignored When writing this field, only legal transitions are valid. See section 14.3.6 Port State Change on page 684.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
PortPhysicalState[b]	RW	4	264	0: No state change 1: Sleep 2: Polling 3: Disabled 4: PortConfigurationTraining 5: LinkUp 6: LinkErrorRecovery 7 - 15: Reserved - ignored When writing this field, only legal transitions are valid. See Volume 2.
LinkDownDefault-State[b]	RW	4	268	0: No state change 1: Sleep 2: Polling 3 - 15: Reserved - ignored When writing this field, only legal transitions are valid. See Volume 2.
M_KeyProtectBits[a]	RW	2	272	See section Section 14.2.4 on page 654.
Reserved	RO	3	274	Reserved, shall be zero.
LMC[c]	RW	3	277	LID mask count for multipath support; its usage is described in 7.11 Subnet Multipathing on page 192.
LinkSpeedActive[b]	RO	4	280	Currently active link speed, indicated as follows: 1: 2.5Gbps 0, 2 - 15: reserved
LinkSpeedEnabled[b]	RW	4	284	Enabled link speed, indicated as follows: 0: No State Change (NOP) 1: 2.5 Gbps 2 - 14: Reserved (Ignored) 15: Set to LinkSpeedSupported value When writing this field, only legal transitions are valid. See Volume 2.
NeighborMTU[b]	RW	4	288	Active maximum MTU enabled on this port for transmit: 1: 256 2: 512 3: 1024 4: 2048 5: 4096 0, 6 - 15: reserved
MasterSMSL[a]	RW	4	292	The administrative SL of the master SM that is managing this port.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
VLCap[b]	RO	4	296	Virtual Lanes supported on this port, indicated as follows: 1: VL0 2: VL0, VL1 3: VL0 - VL3 4: VL0 - VL7 5: VL0 - VL14 0, 6 - 15: reserved
Reserved	RO	4	300	Reserved, shall be zero.
VLHighLimit[b]	RW	8	304	Limit of High Priority component of VL Arbitration Table, as defined in section 7.6.9 VL Arbitration and Prioritization on page 161.
VLArbitrationHighCap[b]	RO	8	312	VL/Weight pairs supported on this port in the VLArbitration table for high priority. Shall be 1 to 64 if more than one data VL is supported on this port, 0 otherwise. See section 7.6.9 VL Arbitration and Prioritization on page 161.
VLArbitrationLowCap[b]	RO	8	320	VL/Weight pairs supported on this port in the VLArbitration table for low priority. Shall be N to 64 if more than one data VL is supported on this port, 0 otherwise, N being the number of data VLs supported. See section 7.6.9 VL Arbitration and Prioritization on page 161.
Reserved	RO	4	328	Reserved, shall be zero.
MTUCap[b]	RO	4	332	Maximum MTU supported by this port. 1: 256 2: 512 3: 1024 4: 2048 5: 4096 0, 6 - 15: reserved
VLStallCount[d]	RW	3	336	Specifies the number of sequential packets dropped that causes the port to enter the VLStalled state. Refer to section 18.2.4.4 Transmitter Queuing on page 858 for details.
HOQLife[c]	RW	5	339	Sets the time a packet can live at the head of a VL queue. Refer to section 18.2.5.4 Transmitter Queueing on page 860 for details.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
OperationalVLs[b]	RW	4	344	Virtual Lanes operational on this port, indicated as follows: 0: No change 1: VL0 2: VL0, VL1 3: VL0 - VL3 4: VL0 - VL7 5: VL0 - VL14 6 - 15: reserved
PartitionEnforce-mentInbound[c]	RW	1	348	Indicates support of optional partition enforcement. If set to one, enables partition enforcement on packets received on this port. Zero disables partition enforcement on packets received from this port.
PartitionEnforce-mentOutbound[c]	RW	1	349	Indicates support of optional partition enforcement. If set to one, enables partition enforcement on packets transmitted from this port. Zero disables partition enforcement on packets transmitted from this port.
FilterRawPacketIn-bound[c]	RW	1	350	Indicates support of optional raw packet enforcement. If set to 1, raw packets arriving on this port are discarded. Zero disables raw enforcement on packets received from this port.
FilterRawPacketOut-bound[c]	RW	1	351	Indicates support of optional raw packet enforcement. If set to 1, raw packets departing on this port are discarded. Zero disables raw enforcement on packets received from this port.
M_KeyViolations[a]	RW	16	352	Counts the number of SMP packets that have been received at this port that have had invalid M_Keys, since power-on or reset. Increments till count reaches all 1s and then must be set back to zero to re-enable incrementing. Setting this component to any value other than zero results in undefined behavior; however, it is recommended that any attempt to set the counter to a non-zero value results in it being left unchanged.
P_KeyViolations[a]	RW	16	368	Counts the number of packets that have been received at this port that have had invalid P_Keys, since power-on or reset. Refer to section 10.9.4 on page 457 for usage description. Increments till count reaches all 1s and then must be set back to zero to re-enable incrementing. Setting this component to any value other than zero results in undefined behavior; however, it is recommended that any attempt to set the counter to a non-zero value results in it being left unchanged.

Table 126 PortInfo

Component	Access	Length(bits)	Offset (bits)	Description
Q_KeyViolations[a]	RW	16	384	Counts the number of packets that have been received at this port that have had invalid Q_Keys, since power-on or reset. See section 10.2.4 on page 403 for usage description. Increments till count reaches all 1s and then must be set back to zero to re-enable incrementing. Setting this component to any value other than zero results in undefined behavior; however, it is recommended that any attempt to set the counter to a non-zero value results in it being left unchanged.
GUIDCap[a]	RO	8	400	Number of GUID entries supported in the GUIDInfo attribute for this port.
Reserved	RO	3	408	Reserved, shall be zero.
SubnetTimeOut[a]	RW	5	411	Specifies the maximum expected subnet propagation delay, which depends upon the configuration of the switches, to reach any other port in the subnet and shall also be used to determine the maximum rate which SubnTraps() can be sent from this port. The duration of time is calculated based on (4.096 uS*$2^{SubnetTimeOut}$).
Reserved	RO	3	416	Reserved, shall be zero.
RespTimeValue[a]	RO	5	419	Specifies the expected maximum time between the port reception of a SMP and the transmission of the associated response. The duration of time is calculated based on (4.096 uS*$2^{RespTimeValue}$).
LocalPhyErrors[b]	RW	4	424	Threshold value. When the count of marginal link errors exceeds this threshold, the local link integrity error shall be detected as described in section 7.12.2 Error Recovery Procedures on page 194.
OverrunErrors[b]	RW	4	428	Threshold value. When the count of buffer overruns over consecutive flow control update periods exceeds this threshold, the excessive buffer overrun error shall be detected as described in section 7.12.2 Error Recovery Procedures on page 194.

a. Applies to channel adapter and router ports and the management port on switches; unused otherwise.
b. Applies to channel adapter and router ports and to all switch ports except the management port; unused otherwise.
c. Applies to channel adapter and router ports. Must be 0 for switch ports.
d. Applies to all switch ports except the management port; unused otherwise.

14.2.5.6.1 INTERPRETATION OF DIAGCODE

The 16-bit *PortInfo:DiagCode* field provides both generic and vendor-specific diagnostic functionality. For all ports, all bits set to zero means the port status is good. Any non-zero value means there are possible error conditions.

The *PortInfo:DiagCode* can provide three levels of diagnostic data:

- A high level, universal status provided by all ports. A *PortInfo:DiagCode* of all zeroes indicates no exception conditions exist on the port. Bits 3-0 of *PortInfo:DiagCode*, when non-zero, have the same meaning for all ports and are defined in Table 127 Standard Encoding of DiagCode Bits 3-0 on page 672 below.
- An optional, high level vendor-specific diagnostic code in bits 14-4 of *PortInfo:DiagCode*. Interpretation of this field requires knowledge of the port diagnostic codes.
- An optional, more detailed vendor-specific port attribute pointed to by *PortInfo:DiagCode*. Availability of this information is indicated by bit 15 of *PortInfo:DiagCode* and the pertinent port attribute is then pointed to by bits 14-4.

Figure 161 DiagCode Fields on page 672 summarizes the structure and interpretation of *PortInfo:DiagCode* fields.

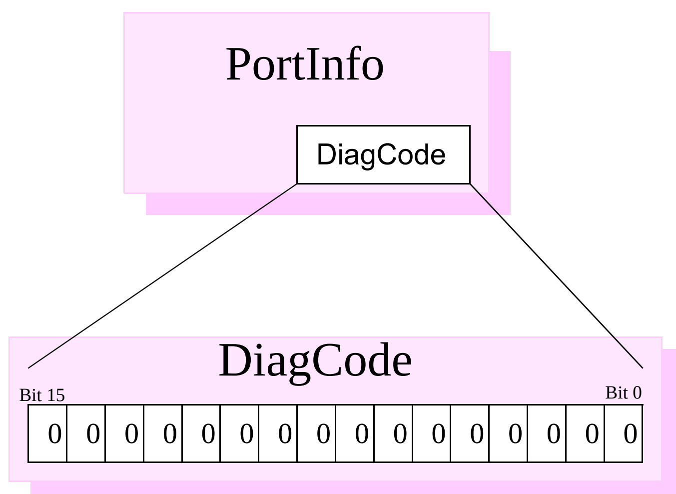


Figure 161 DiagCode Fields

Bits 3-0 are reserved for diagnostic codes common to all ports.

Bits 14-4 provide vendor-specific diagnostic codes.

Bit 15 provides a means to chain vendor diagnostic attributes

The error information in bits 3-0 is interpreted in a standard fashion for all ports as shown in Table 127 Standard Encoding of DiagCode Bits 3-0 on page 672.

Table 127 Standard Encoding of DiagCode Bits 3-0

DiagCode Bits 3-0	Description
0x0	Port Ready
0x1	Performing Self Test

Table 127 Standard Encoding of DiagCode Bits 3-0

DiagCode Bits 3-0	Description
0x2	Initializing
0x3	Soft Error - Port Has A Non-Fatal Error And May Be Used
0x4	Hard Error - Port May Not Be Used
0x5 - 0xF	Reserved

Bits 14-4 of *PortInfo:DiagCode* are used for vendor-specific modifiers to the standard diagnostic information, as shown in Figure 162 DiagCode Bits on page 673.

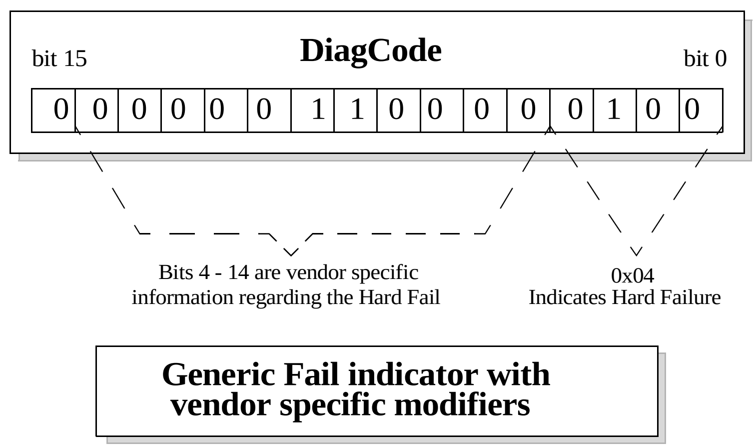


Figure 162 DiagCode Bits

If bit 15, the *IndexForward* bit, is zero (0), bits 14-4 of the *PortInfo:DiagCode* represent a vendor-specific diagnostic code. Interpretation of the returned information is outside the scope of this specification. Further diagnostic information might be provided in known locations in one or more vendor-specific Attributes.

If the *IndexForward* bit is set, bits 14-4 of the *PortInfo:DiagCode* field are used to index into the *VendorDiag* Attribute data for vendor-specific diagnostic information. This allows dynamic chaining of diagnostic information based on the type of exception. Bits 14-4 are interpreted as an Attribute Modifier to be specified with an *SubnGet(VendorDiag)* to the port being examined as defined in section 14.2.5.14 VendorDiag on page 679.

14.2.5.7 P_KEYTABLE

The P_KeyTable Attribute provides the means for assigning the P_Keys for ports.

The Attribute Modifier is divided in two halves:

- The least significant 16 bits are a pointer to a block of 32 P_Key entries to which this Attribute applies. Valid values are 0 - 1023, and are further limited by the size of the P_Key table (specified by the *PartitionCap* for CAs, routers, and switch management ports or *PartitionEnforcementCap* for external ports on switches) for that node and any entries in the block beyond the end of the table are read-only and set to 0.
- For switches, the upper 16 bits select the switch port, where valid values are 1 - 255 to select physical ports and zero to select the switch management port.

  For CA and router, the upper 16 bits are ignored and the operation is performed on the port that received the SMP.

Table 128 P_KeyTable

Component	Access	Length (bits)	Description
P_KeyTable Block	RW	512	List of 32 P_Key Block Elements.

Table 129 P_Key Block Element

Component	Length (bits)	Description
Membership-Type	1	If set to zero, the P_Key is *limited* type and the endnode may accept a packet with a matching *full* P_Key, but may not accept a packets with a matching *limited* P_Key. If set to one, the P_Key is *full* type and the endnode may receive packets with matching *full* or *limited* P_Key. A full description is in section 10.9.1.1 on page 455.
P_KeyBase	15	Base value of the P_Key that the endnode will use to check against incoming packets.

14.2.5.8 SLTOVLMAPPINGTABLE

The SLtoVLMappingTable Attribute provides the means for setting the SL to VL Mapping of a switch, CA, and router and its usage is described in 7.6.6 VL Mapping Within a Subnet on page 159.

For a switch, this attribute is specific to an input port / output port combination to which the specific SL to VL mapping applies:

- bits 15-8 of the Attribute Modifier specify the input port which can be 1 to N, where N selects the physical port or 0 to indicate that the input port is the management port.
- bits 7-0 of the Attribute Modifier specify the output port which can be 1 to N, where N selects the physical port.
- bits 31-16 must be zero.

For CA and router, this attribute corresponds to the port receiving the SMP (the Attribute Modifier is ignored).

Table 130 SLtoVLMappingTable

Component	Access	Length(bits)	Offset (bits)	Description
SL0toVL	RW	4	0	The number of the VL on which packets using SL0 are output. 15 forces the packets to be dropped.
SL1toVL	RW	4	4	The VL associated with SL1
SL2toVL	RW	4	8	The VL associated with SL2
SL3toVL	RW	4	12	The VL associated with SL3
SL4toVL	RW	4	16	The VL associated with SL4
SL5toVL	RW	4	20	The VL associated with SL5
SL6toVL	RW	4	24	The VL associated with SL6
SL7toVL	RW	4	28	The VL associated with SL7
SL8toVL	RW	4	32	The VL associated with SL8
SL9toVL	RW	4	36	The VL associated with SL9
SL10toVL	RW	4	40	The VL associated with SL10
SL10toVL	RW	4	44	The VL associated with SL11
SL12toVL	RW	4	48	The VL associated with SL12
SL13toVL	RW	4	52	The VL associated with SL13
SL14toVL	RW	4	56	The VL associated with SL14
SL15toVL	RW	4	60	The VL associated with SL15

14.2.5.9 VLARBITRATIONTABLE

The VLArbitrationTable Attribute provides the means for setting the VL Arbitration for ports on CA, routers and switches and its usage is described in 7.6.9 VL Arbitration and Prioritization on page 161 (xref to section 7.6.9).

The Attribute Modifier is divided in two halves. The upper 16 bits specify the part of the tables that is accessed.

- 1 - lower 32 entries of the low priority VL Arbitration Table.
- 2 - upper 32 entries of the low priority VL Arbitration Table.
- 3 - lower 32 entries of the high priority VL Arbitration Table.
- 4 - upper 32 entries of the high priority VL Arbitration Table.
- 0, 5-65535 - reserved.

For switches, the least significant 16 bits of the attribute modifier specify the external port in bits 7-0 and bits 15-8 are reserved and must be set to 0.

For CA and router, this attribute corresponds to the port receiving the SMP (the lower 16 bits of the Attribute Modifier is ignored).

Table 131 VLArbitrationTable

Component	Access	Length (bits)	Offset (bits)	Description
VL/Weight pairs	RW	512	0	Lists of 32 VL/Weight Block elements, for which there may be up to 64 in total for a given priority. The interpretation is as follows: 1 - values 0 -31 of low priority 2 - values 32 -63 of low priority 3 - values 0 - 31 of high priority 4 - values 32 -63 of high priority

Table 132 VL/Weight Block Element

Component	Length (bits)	Offset (bits)	Description
reserved	4	0	
VL	4	4	VL associated with element.
Weight	8	8	Weight associated with element, as defined in section 7.6.9 VL Arbitration and Prioritization on page 161, zero indicates that this element is skipped.

14.2.5.10 LINEARFORWARDINGTABLE

The LinearForwardingTable Attribute provides the means for setting the linear forwarding table of a switch for the Unicast LIDs.

The Attribute Modifier is a pointer to a block of 64 LIDs to which this attribute applies. Valid values are from 0 to 767, and are further limited by the size of the Linear Forwarding Table of the switch. Any entries in the block beyond the end of the table are ignored on write and read back as zero. If an invalid port number is written into an entry, packets sent to this

LID will be discarded and that entry shall be read back as 0xFF to indicate that an invalid port number was used.

Table 133 LinearForwardingTable

Component	Access	Length(bits)	Description
LinearFor-wardingTable Block	RW	512	List of 64 Port Block Elements.

Table 134 Port Block Element

Component	Length(bits)	Description
Port	8	Port to which packets with the LID corresponding to this entry are to be forwarded.

14.2.5.11 RANDOMFORWARDINGTABLE

The RandomForwardingTable Attribute provides the means for setting the random forwarding table of a switch for the Unicast LIDs.

The Attribute Modifier is a pointer to a block of 16 LID/port pairs to which this Attribute applies. Valid values are from 0 to 3071, and are further limited by the size of the Random Forwarding Table of the switch and any entries in the block beyond the end of the table are read-only and set to 0.

Table 135 RandomForwardingTable

Component	Access	Length(bits)	Description
RandomFor-wardingTable Block	RW	512	List of 16 LID/Port Block Elements.

Table 136 LID/Port Block Element

Component	Length(bits)	Offset (bits)	Description
LID	16	0	Base LID.
Valid	1	16	This LID/Port pair is valid. Note that setting this parameter to 0 allows the removal of entries.
LMC	3	17	the LMC of this LID.
Reserved	4	20	
Port	8	24	Port to which packets with this LID/LMC corresponding to this entry are to be forwarded.

14.2.5.12 MULTICASTFORWARDINGTABLE

This MulticastForwardingTable Attribute provides the means for setting the multicast forwarding table of a switch.

The ten low-order bits of the Attribute Modifier are a pointer to a block of 32 PortMask entries to which this attribute applies. Valid values are from 0 to 511, and are further limited by the size of the Multicast Forwarding Table of the switch. Any entries in the block beyond the end of the table are read-only and set to 0.

The four high-order bits of the Attribute Modifier indicate the position (p) of the 16-bit PortMask entry of this Attribute. Each PortMask entry specifies only 16 bits of the 256 possible bits of a port mask of a maximum size switch. The remaining 18 bits of the Attribute Modifier shall be set to zero.

Table 137 MulticastForwardingTable

Component	Access	Length(bits)	Description
MulticastForwardingTable Block	RW	512	List of 32 PortMask Block Elements.

.

Table 138 PortMask Block Element

Component	Length(bits)	Description
PortMask	16	16 bits starting at position 16*p of the port mask associated with the particular LID. An incoming packet with this LID is forwarded to all ports for which the bit in the port mask is set to 1. Note that an invalid LID is indicated with an all zero PortMask.

14.2.5.13 SMINFO

The SMInfo attribute is used by Subnet Managers to exchange information during subnet discovery and polling as described in section 14.4 Subnet Manager on page 687. This attribute shall be available on a port where a Subnet Manager resides.

Table 139 SMInfo

Component	Access	Length (bits)	Offset (bits)	Description
GUID	RO	64	0	PortGUID of the port where the SM resides.
SM_Key	RO	64	64	Key of this SM. This is shown as 0 unless the requesting SM is proven to be the master, or the requester is otherwise authenticated.

Table 139 SMInfo

Component	Access	Length (bits)	Offset (bits)	Description
ActCount	RO	32	128	Counter that increments each time the SM issues an SMP or performs other management activities. Used as a “heartbeat” indicator by standby SMs.
Priority	RO	4	160	Administratively assigned priority for this SM. Can be reset by master SM. Zero is lowest priority. An out-of-band mechanism for setting this value must be provided. The default value, if not set by the out-of-band mechanism, shall be zero.
SMState	RO	4	164	Enumerated value indicating this SM’s state. Enumerated as follows: 0 - not active 1 - discovering 2 - standby 3 - master 4-15 - Reserved

14.2.5.14 VENDORDIAG

The VendorDiag Attribute provides a way to obtain vendor specific diagnostic information. The interpretation of the *VendorDiag:DiagData* is specific to the port in question. It is accessible from ports on CAs, routers, and the management port on a switch.

Table 140 VendorDiag

Component	Access	Length (bits)	Offset (bits)	Description
NextIndex	RO	16	0	Next Attribute Modifier to get to diagnostic Info. Set to zero if this is last or only diagnostic data.
DiagData	RO	496	16	Vendor specific diagnostic information. Format is undefined.

Section 14.2.5.6.1 Interpretation of DiagCode on page 671 describes the use of the *PortInfo:DiagCode* forwarding mechanism used to obtain the address modifier for the VendorDiag attribute during interpretation of diagnostic codes. An example of the use of the *IndexForward* bit, bit 15 of the

PortInfo:DiagCode component, is shown in Figure 163 Index Forwarded Diagnostic Information on page 680.

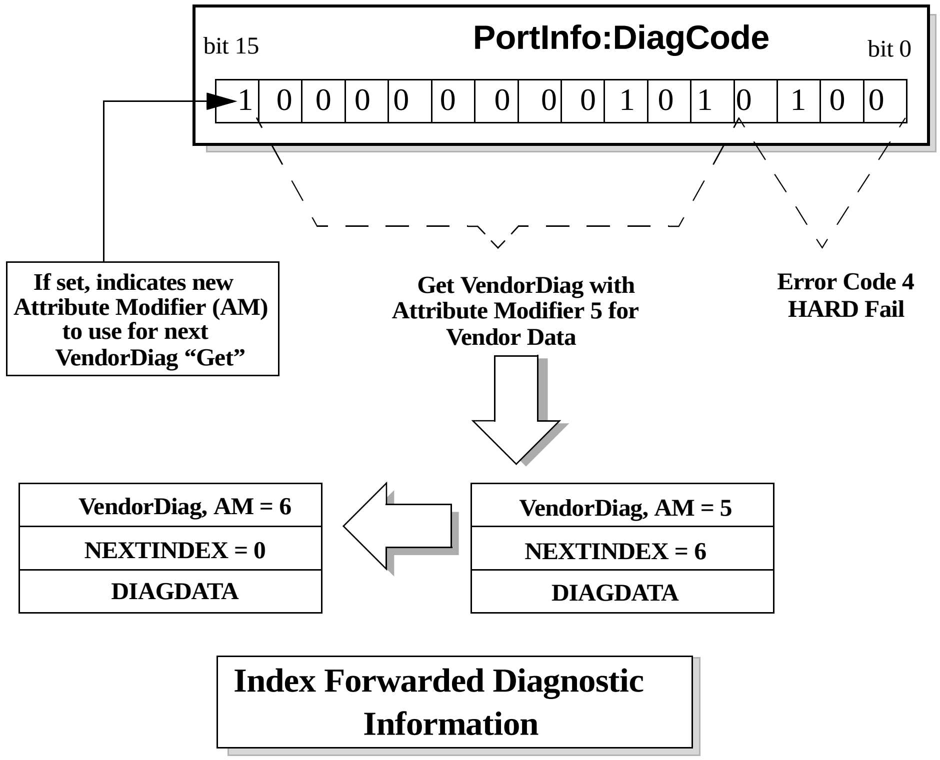


Figure 163 Index Forwarded Diagnostic Information

In the above example the *PortInfo:DiagCode* with the *IndexForward* bit set indicates that VendorDiag Attribute Modifier 5 of this port contains vendor-specific diagnostic information. When VendorDiag Attribute Modifier 5 is retrieved, the *VendorDiag:NextIndex* value indicates more data at Attribute Modifier 6. The retrieval of Attribute Modifier 6 returns a *VendorDiag:NextIndex* of 0, indicating the end of the diagnostic data.

14.2.5.15 LedInfo

The LedInfo Attribute provides the ability to turn on or off a LED optionally provided by a CA, router, and switch using SMPs. This LED is not specified and the implementation of this LED is vendor-specific. It has no association with LEDs that are specified by this or other volumes of the IB specification. A CA, router, and switch shall indicate its support of this attribute in the *PortInfo:CababilityMask*.

Table 141 LedInfo

Component	Type	Length (bits)	Offset (bits)	Description
LedMask	RW	1	0	Set to 1 for LED on, and 0 for LED off. The response packet shall indicate actual LED state.
Reserved	RO	31	1	

14.3 SUBNET MANAGEMENT AGENT

Each CA and router, and switch will have a Subnet Management Agent (SMA) that communicates with the SMI and SM as described in section 13.3.2 Required Managers and Agents on page 602. The SMA will respond and generate SMPs as described in Table 108 SM MAD Sources and Destinations on page 636. This section describes the detailed requirements of SMA behavior where the operations defined below assume the receipt of a valid SMP. A SMP is valid if it satisfies all applicable validation checks as specified in Section 13.5.3 MAD Validation on page 635.

14.3.1 SUBNGET

A SMA may receive a SMP from the subnet containing a SubnGet at any time. The requester, the master SM, will fill the *MADHeader:M_Key* field of the SMP header with a M_Key that matches the value of the M_Key of the port corresponding to the receiving SMA if it expects the receiving SMA to check it.

A SMP containing a SubnGetResp is returned according to the rules in section 14.3.3 SubnGetResp on page 683.

14.3.2 SUBNSET

An SMA may receive a SMP from the subnet containing a *SubnSet* at any time. The requester, the master SM, will fill the *MADHeader:M_Key* field of the SMP header with a M_Key that matches the value of the M_Key of the port corresponding to the receiving SMA if it expects the receiving SMA to check it.

C14-25: If the *PortInfo:M_Key* component is zero, the SMA shall update the appropriate components with the contents of the attribute contained in the SMP.

C14-26: If the *PortInfo:M_Key* component is non-zero and M_Key matching, if required, is successful according to the rules specified in section 14.2.4 Management Key on page 654, the SMA shall update the appropriate components with the contents of the attribute contained in the SMP.

C14-27: The SMA shall ignore requests to change non-settable (RO) components of attributes.

A SMP containing a SubnGetResp is returned according to the rules in section 14.3.3 SubnGetResp on page 683.

14.3.3 SubnGetResp

C14-28: When the SMA receives a validated SubnGet or SubnSet and the *PortInfo:M_Key* component is zero, then the SMA shall generate a *SubnGetResp.*

C14-29: When the SMA receives a validated SubnGet or SubnSet and the *PortInfo:M_Key* component is non-zero and M_Key matching, if required, is successful according to the rules specified in section 14.2.4 Management Key on page 654, then the SMA shall generate a *SubnGetResp*, otherwise the request is silently discarded.

C14-30: If the SMA generates a *SubnGetResp*, it shall fill the attribute identified in the request with the appropriate contents of component information.

C14-31: If the SMA generates a *SubnGetResp*, it shall use the *MADHeader:TransactionID* obtained from the request SMP in the response SMP.

C14-32: This compliance statement is obsolete and has been removed.

If the SMA generates a SubnGetResp, the content of MAD-Header:M_Key in the SMP header is undefined. It could, for example, be copied without change from the request or zeroed out. If the SMA generates a *SubnGetResp*, it should send the SMP containing the *SubnGetResp* in less than *PortInfo:RespTimeValue* of the receiving port, where requirements for response time are described in section 13.4.6.2 Timers and Timeouts on page 613.

After transmission of the response, the SMA discards any residual state associated with that SMP.

14.3.4 SubnTrap

Traps may be issued by any port on the subnet. Ports that support this mechanism will indicate this by setting the *PortInfo:CapabilityMask:IsTrapSupported* bit.

o14-1: If the SMA generates a *SubnTrap*, it shall fill the M_Key field of the SMP with zero.

o14-2: If the SMA generates a sequence of traps, the interval between successive traps shall not be smaller than the subnet timeout, which is specified by the *PortInfo:SubnetTimeOut* component.

This mechanism is used to limit the number of traps sent on the subnet.

o14-3: If the SMA generates a trap, it shall only send it when the *PortInfo:Portstate* is Active.

o14-3.a1: If the SMA generates a trap, it shall set the source LID to the PortInfo:LID of the originating port.

This section describes the application of the architected traps for subnet management event reporting. The entire list of subnet management class traps are described in section 14.2.5.1 Notices and Traps on page 660.

14.3.5 SUBNTRAPREPRESS

An SMA may receive a SMP from the subnet containing a *SubnTrapRepress* in reaction to a *SubnTrap* sent by the SMA itself. The requester, the master SM, will fill the *MADHeader:M_Key* field of the SMP header with a M_Key that matches the value of the M_Key of the port corresponding to the receiving SMA if it expects the receiving SMA to check it.

o14-3.a1: If the *PortInfo:M_Key* component is zero, the SMA shall process the SubnTrapRepress according to 13.4.9 Traps on page 624.

o14-3.a2: If the *PortInfo:M_Key* component is non-zero and M_Key matching, if required, is successful according to the rules specified in section 14.2.4 Management Key on page 654, the SMA shall process the SubnTrapRepress according to 13.4.9 Traps on page 624.

o14-3.a3: The SMA shall not send any message in response to a *SubnTrapRepress* message.

14.3.6 PORT STATE CHANGE

Switches are capable of reporting port state changes.

o14-4: If required to send a trap or log a notice, the SMA residing on the management port of a switch shall monitor the *SwitchInfo:PortStateChange* bit for a transition from zero to one.

o14-5: If the management port supports Traps as indicated in the *PortInfo:CapabilityMask.IsTrapSupported*, the SMA shall send a trap 128 to the SM indicated by the *PortInfo:MasterSMLID* for a port state change on a switch.

o14-6: If the management port supports Notices as indicated the *PortInfo:CapabilityMask.IsNoticeSupported*, the SMA shall log a notice for a port state change on a switch.

The contents of the trap or notice is filled with information from Table 117 Notice DataDetails For Trap 128 on page 660.

14.3.7 TRANSPORT KEY MISMATCH

Transport key mismatch happens when a key residing in the headers of an incoming packet does not match the key for the destination QP during packet validation as described in the section 9.6 Packet Transport Header Validation on page 236 of the transport chapter.

C14-33: The SMA shall monitor P_Key and Q_Key mismatches detected by the transport services on that port.

C14-34: If a P_Key or Q_Key mismatch occurs, the SMA shall report the current count via the contents of *PortInfo:P_KeyViolations* or *PortInfo:Q_KeyViolations* components of the PortInfo attribute (see section 14.2.5.6 PortInfo on page 665).

o14-7: If the port supports Traps as indicated in the *PortInfo:CapabilityMask.IsTrapSupported*, the SMA shall send a trap 257 or 258 to the SM indicated by the *PortInfo:MasterSMLID* for P_Key and Q_Key mismatches, respectively.

o14-8: If the port supports Notices as indicated the *PortInfo:CapabilityMask.IsNoticeSupported*, the SMA shall log a notice for P_Key and Q_Key mismatches.

The contents of the trap or notice is filled with information from Table 120 Notice DataDetails For Traps 257 and 258 on page 661 for P_Key and Q_Key mismatches, respectively.

14.3.8 M_KEY MISMATCH

As a result of the M_Key residing in the SMP header, the SMA is responsible for checking it. The SMA will perform an M_Key check as described in section 14.2.4.1 Levels of Protection on page 655. If the check fails and a lease period countdown is not already in effect, the SMA starts a lease period countdown as described in section 14.2.4.2 Lease Period on page 656.

o14-9: If the port supports Traps as indicated in the *PortInfo:CapabilityMask.IsTrapSupported*, the SMA shall send a trap 256 to the SM indicated by the *PortInfo:MasterSMLID* when a M_Key mismatch is detected.

o14-10: If the port supports Notices as indicated the *PortInfo:CapabilityMask.IsNoticeSupported*, the SMA shall log a notice when a M_Key mismatch is detected.

The contents of the trap or notice is filled with information from Table 119 Notice DataDetails For Trap 256 on page 661.

14.3.9 LINK LAYER ERRORS

The link layer performs error detection and recovery as described in section 7.12 Error detection and handling on page 192. The SMA is responsible for monitoring the Local link integrity, excessive buffer overrun, and flow control update counters of the link.

o14-11: When a Local Link Integrity error occurs on a port that supports Traps (*PortInfo:CapabilityMask.IsTrapSupported* is set), the port's SMA shall send Trap 129 to the SM at the address contained in *PortInfo:MasterSMLID*; when an Excessive Buffer Overrun error occurs, Trap 130 shall be sent; and when a Flow Control Update error occurs, Trap 131 shall be sent.

o14-12: If the port supports Notices as indicated the *PortInfo:CapabilityMask.IsNoticeSupported*, the SMA shall log a notice when the Local link integrity, excessive buffer overrun, or flow control update counters increment.

The contents of the trap or notice is filled with information from Table 118 Notice DataDetails For Traps 129, 130 and 131 on page 661 for Local link integrity, excessive buffer overrun, and flow control update counter changes, respectively.

14.4 SUBNET MANAGER

There may be one or more Subnet Managers operating on a subnet as described in section 13.3.2 Required Managers and Agents on page 602. Each Subnet Manager (SM) indicates its presence on the subnet by setting the *IsSM* bit in the *PortInfo:CapabilityMask* on the port where it resides (see Table 126 PortInfo on page 665). There may be several SMs on a particular node, each residing on different subnets.

C14-35: An SM shall always be associated with one port and one subnet.

Each SM is always in a particular state: Master, Standby, Discovering or Not-active.

The algorithm used to initialize the subnet, the algorithm for adding/deleting routes in response to subnet changes, the mechanisms for failover from master SM to standby SM, and the mechanism for transfer of mastership from master SM to standby SM are beyond the scope of the specification. However, there are mechanisms specified in this section that may be used to support these operations.

C14-36: A SM shall comply with the state machine shown in Figure 164 SMInfo State Transitions on page 688 during its startup and shall become either a master or standby SM.

Correct execution of the state machine ensures that there be only one Master SM on a subnet at any time and that after startup, a SM becomes either a Standby or Master on the subnet.

Furthermore, the state machine specifies how a single Master SM is maintained during subnet topology changes, packet loss, addition/removal of SMs, and subnet mergers. Subsequent sections include the specification of optional mechanism that may be used by SMs to communicate and a description of some SM operations on the subnet, but none of these are required for SM compliance.

14.4.1 SM STATE MACHINE

The behavior of the SM is specified in terms of the SM state machine. This section starts by defining the specific mechanisms used by the SM: the *SMInfo* attribute, control packets that SMs may exchange, a set of timers, and the exception conditions reported to the higher layer (administrator).

Each SM provides a SMInfo attribute that is specified in Table 139 SMInfo on page 678 and is exported from the port where it resides.

C14-37: The *SMInfo:Priority* and *SMInfo:SM_Key* shall be configurable through an out-of-band mechanism that is outside the scope of this specification.

The contents of the components in the *SMInfo* attribute determine which SM in a multi-SM subnet becomes Master: the one with the highest Priority and the lowest GUID.

Each Standby SM should be ready to become Master when the current Master fails (or gets disconnected). Also, mastership will be handed over when the Master detects another SM with a higher Priority (or same Priority and lower GUID), e.g., during merger of two subnets. Handover takes place only between SMs that have the right SM_Key.

Under certain circumstances, e.g., when the number of Standby SMs becomes an obstacle to scaleability, then a Master SM may force other SMs to become Not-active.

C14-38: In order to assure interoperability, each SM shall respond to *SubnGet(SMInfo)* or *SubnSet(SMInfo)* with a *SubnGetResp(SMInfo)*.

Figure 164 SMInfo State Transitions on page 688 summarizes the states that a SM may represent in the *SMInfo:SMState.*

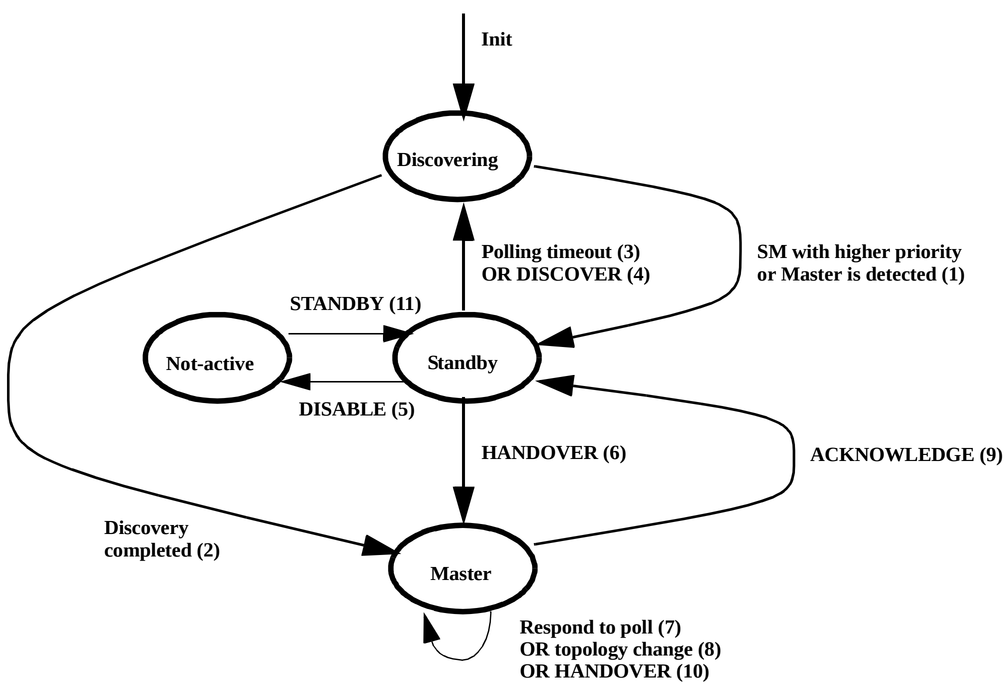


Figure 164 SMInfo State

The state transitions correspond to the event numbers in the text below. The following sections describe the behavior of the SM in each of the states and the (externally driven) events that cause state changes.

14.4.1.1 CONTROL PACKETS

Control packets may be exchanged between SMs using a SMP that contains a *SubnSet(SMInfo)* where the *MADHeader:AttributeModifier* is used to select from one of the following actions specified in Table 142 SM Control Packets on page 689. The SM is not required to generate these control packets and may use mechanisms that are beyond the scope of the specification to implement similar functions, however, a SM is required to correctly respond to them.

Table 142 SM Control Packets

MADHeader:AttributeModifier	Description
1	HANDOVER: Is used to initiate the process of handing over Mastership to a higher priority Standby SM or Master.
2	ACKNOWLEDGE: Is used to acknowledge the handover
3	DISABLE: Is used to disable a Standby SM.
4	STANDBY: Is used to return a Not-active SM to Standby.
5	DISCOVER: Causes a Standby SM to go to Discovering.

14.4.1.2 DISCOVERING STATE

DISCOVERING is the initial state.

C14-39: At startup, a SM shall enter the DISCOVERING state.

C14-40: In the DISCOVERING state, the SM shall perform repetitive *SubnGet(*)* to find all nodes and SMs on the subnet.

Section 14.4.2 Subnet Discovery Actions on page 693 summaries many of the attributes that are collected during discovery. The SM will typically use direct-routed SMPs to reach all the endnodes. The sequence of discovery is implementation specific and beyond the scope of the specification.

C14-41: If the SM in the DISCOVERING state finds another SM with a higher Priority than its own, or with the same Priority and a lower GUID, or with a *SMInfo:SMState* = MASTER, then the SM shall yield and change its *SMInfo:SMState* to STANDBY.

See Figure 164 SMInfo State Transitions on page 688, number 1. At this point the SM stops the discovery and starts operating as a Standby SM.

C14-42: If the SM in the DISCOVERING state completes the discovery process without finding a Master or a higher priority (lower GUID) SM, it shall assume the role of a Master by changing its *SMInfo:State* to MASTER.

C14-43: The master SM shall initially send to all the nodes on the subnet *SubnSet(PortInfo)* SMPs with *MasterSMLID* and *MasterSMSL* that specify a path to itself.

See Figure 164 SMInfo State Transitions on page 688, number 2.

C14-44: If the SM discovers that it does not have a M_Key required to configure a CA, switch, or router on the subnet it shall notify the higher-layer (through an interface beyond the scope of the specification).

14.4.1.3 STANDBY STATE

C14-45: Standby SMs shall not configure the subnet.

C14-46: Each Standby SM shall poll the Master SM with *SubnGet(SMInfo)* SMPs, addressed to its *PortInfo:MasterSMLID.* As long as the Standby determines that the Master is alive, it stays in *SMInfo:SMState* = STANDBY.

The minimum interval between polling is set by the higher-layer (through an interface beyond the scope of the specification). The actual interval may be longer for Standby SMs with lower Priority or when there is a larger number of Standby SMs on the subnet. The actual polling interval is installation specific and is not specified in the architecture. The Master may use the optional control packets to disable Standby SMs if it determines that there is excessive polling in the subnet.

C14-47: If the Standby SM does not receive a *SubnGetResp(SMInfo)* that indicates progress in the ActCount, within the number of retries that is set by the higher-layer (through an interface beyond the scope of the specification), then it should conclude that the Master is no longer alive (or accessible) and it shall change its *SMInfo:SMState* back to DISCOVERING.

See Figure 164 SMInfo State Transitions on page 688, number 3.

C14-48: If a Standby SM receives a DISCOVER packet, i.e. a *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 5, then it shall change its *SMInfo:SMState* to DISCOVERING.

See Figure 164 SMInfo State Transitions on page 688, number 4.

C14-49: If a Standby SM receives a DISABLE packet, i.e., a *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 3, then it shall change its *SMInfo:SMState* to NOT-ACTIVE.

See Figure 164 SMInfo State Transitions on page 688, number 5. This allows the Master to disable Standby SMs if it determines that the amount of polling creates a scaleability problem.

The Master SM may relinquish mastership of the subnet to a Standby with higher priority and correct SM_Key if it detects one. Event 6 specifies the Standby behavior during that transfer. The Master's behavior is specified in section 14.4.1.5 Master State on page 692.

C14-50: If a Standby SM receives a HANDOVER control packet, i.e., a *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 1, it shall return a *SubnGetResp(SMInfo)*.

The steps necessary to take control of the subnet are beyond the scope of specification. However, the standby SM may, for example, perform the following operations:

1) The standby SM receiving the HANDOVER control packet obtains necessary topology information, possibly obtaining data defined as required record attributes specified in section 15.2.5.1 Summary of Attributes on page 709 from subnet administration residing with the current Master SM.
2) The standby SM should send to all the nodes on the subnet a *SubnSet(PortInfo)* with MasterSMLID and MasterSMSL that specify a path to itself.
3) The standby SM may send the current Master SM an ACKNOWLEDGE control packet, i.e. an *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 2.
4) If the standby SM does not receive a *SubnGetResp(SMInfo)*, it should notify the higher-layer (through an interface beyond the scope of the specification). This is an indication that the Master may have died in the middle of an unsuccessful hand over.
5) After receiving a *SubnGetResp(SMInfo)*, it assumes the role of a Master by changing its *SMInfo:State* to MASTER. See Figure 164 SMInfo State Transitions on page 688, number 6.

14.4.1.4 NOT-ACTIVE STATE

C14-51: If a SM is in the NOT-ACTIVE state, it shall indicate this by setting the *SMInfo:SMState* to NOT-ACTIVE.

C14-52: If the SM is in the NOT-ACTIVE state, it shall not send *SubnSet()* or *SubnGet()* SMPs.

C14-53: If the SM is in the NOT-ACTIVE state, it shall respond to *SubnSet(SMInfo)* and *SubnGet(SMInfo)* SMPs.

C14-54: If the SM in the NOT-ACTIVE state and it receives a STANDBY packet, i.e., a *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 5, it shall change its state to STANDBY.

See Figure 164 SMInfo State Transitions on page 688, number 11.

14.4.1.5 MASTER STATE

The Master starts its operation by topology discovery, LID verification and assignment (if applicable), path verification and calculation, etc. (as specified in section 14.4.3 Initialization Actions on page 694).

C14-55: Only the Master SM shall configure subnet nodes.

C14-56: The Master SM shall perform periodic sweeps of the subnet to check for changes in general, and for the appearance of new SMs, in particular.

C14-57: If the M_Key protection mechanism, as described in 14.2.4.1 Levels of Protection on page 655, is being used, the Master SM shall sweep the subnet at a rate that will refresh the lease period of every port on the subnet.

Section 14.4.5 Subnet Sweeping on page 699 describes the sweep activities.

C14-58: The Master shall increment the *SMInfo.ActCount* every time it performs a management operation or issues an SMP.

When the SM in Master state receives a valid *SubnGet(SMInfo)* or *SubnSet(SMInfo)*, it should respond with a *SubnGetResp(SMInfo)* when M_Key matching, if required, is successful as described in section 14.4.6 Authentication on page 699. This is required in order to support the Standby polling mechanism.

C14-59: If during the sweep the Master detects a topology change, then it must perform the operations listed below:

- If the change is a link going down, then the Master needs to possibly establish new paths and send new MasterSMLID/SLs to the affected nodes. The details are beyond the scope of the specification.
- If the Master detects a new link, then it starts discovering the subnet beyond the new links, using (partially) direct routed SMPs.

If the SM discovers that it does not have a M_Key required to configure a CA, switch, or router on the subnet, it will notify the higher-layer (through an interface beyond the scope of the specification).

C14-60: If during the discovery a Master SM finds another Master SM with lower priority (or same priority and higher GUID), it shall stop the discovery, waiting for the other Master to relinquish control of its portion of the subnet.

C14-61: If during the discovery the Master SM finds a SM that has a higher priority (or same priority and lower GUID) and it has the appropriate SM_Key, it shall complete the discovery in order to determine the highest priority SM (with an appropriate SM_Key) in the new part of the subnet (if applicable) and it shall relinquish control of its portion of the subnet to that SM.

The steps necessary to transfer control of the subnet from one master SM to another is beyond the scope of specification, however, the master SM may use the optional control packets to perform the handover process as follows:

- It may complete operations in progress.
- It sends the higher priority SM a HANDOVER packet, i.e. a *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 1.
- It continues responding to polls from Standby SMs until it receives an ACKNOWLEDGE packet, i.e., a *SubnSet(SMInfo)* with *MADHeader:AttributeModifier* set to the value of 2 from the higher priority SM.
- When it receives an ACKNOWLEDGE packet, it will change its *SMInfo:SMState* to STANDBY and return a *SubnGetResp(SMInfo)*. See Figure 164 SMInfo State Transitions on page 688, number 9.
- If it does not receive an ACKNOWLEDGE packet, then it informs the higher-layer (through an interface beyond the scope of the specification).

If a Master SM discovers a higher priority Master SM does not have the proper SM_Key, then it should not relinquish mastership of its portion of the subnet and it should report to the higher-layer (through an interface beyond the scope of the specification) that it discovered another Master SM on the same subnet.

14.4.2 SUBNET DISCOVERY ACTIONS

The SM collects information from the attributes and records them for later use during configuration of the subnet. The discovery algorithm is outside the scope of the specification, however, discovery may consist of:

- probing the subnet with directed route packets
- loading a topology database from persistent storage
- a combination of information that is loaded from persistent storage and obtained by probing subnet nodes

During discovery, the SM scans the attributes described in section 14.2.5 Attributes on page 658 to obtain information not limited to the following:

- VLs on each Port
- MTU of the Port
- Link Width on each Port
- Link Speed on each Port
- Physical topology, connectivity of links between nodes
- P_Key table sizes
- GUID table sizes
- support for various capabilities
- device type: switch, CA, or router
- power-on diagnostic status
- for switches,
    - size of switch linear-forwarding or random-forwarding tables
    - support for multicast forwarding table and size
    - presence of the optional VL arbitration table
    - presence of the optional SL-to-VL mapping table

14.4.3 INITIALIZATION ACTIONS

The algorithms and policy that are necessary to set many of the subnet attributes is outside the scope of the specification,. However, there is a core set of attributes that the SM is responsible for setting in order to make the subnet functional.

C14-62: The Master SM shall initialize the subnet components specified in the following Table 143 Initialization on page 695.

Table 143 Initialization

Component	Description
PortInfo:LID	The SM shall assign a unicast LID address to each endport on the subnet. LID usage is described in section 4.1.2 Channel Adapter, Switch, and Router Addressing Rules on page 121 and 4.1.3 Local Identifiers on page 121. The SM shall also set up the forwarding tables of the switches in the subnet such that each base LID of a port on a CA, Router, or switch port 0 is reachable from any other port on the subnet within the boundaries set by partitions. The LID ranges assigned by a SM may be further limited by the Range Record described in section 15.2.5.16 RangeRecord on page 717.
PortInfo:LMC	The SM shall assign an LMC for each CA and router port on the subnet. LMC usage is described in section 4.1.3 Local Identifiers on page 121. The SM may program the LMC on a port to any value between 0 and 7, to allow use of multiple LIDs (1-128) in addressing the port. The SM shall not assign overlapping ranges of LIDs based on LMCs to different ports.
PortInfo:GidPrefix	The SM shall assign a Subnet Prefix for the subnet based on the presence of a router and the rules specified in section 4.1.3 Local Identifiers on page 121.
PortInfo:OperationalVL	The SM shall initialize the VL tables for CAs, switches and routers. The SM will examine the supported VLs in the *PortInfo:VLCap* at both ends of every link and sets the maximum number of VLs by setting the *PortInfo:OperationalVL* at each end to the smaller of the two supported number of VLs. The description of VL initialization resides in section 7.6.7 Initialization and Configuration on page 161.
PortInfo:NeighborMTU	The SM shall initialize the port MTU for CAs, switches and routers. The SM examines the supported MTU size in the *PortInfo:MTUCap* at both ends of every link and sets the maximum MTU parameter on the ports in the *PortInfo.NeighborMTU* at each end to the smaller of the two supported size. When a node powers on, it will set the MTU to 256 bytes.
PortInfo:SubnetTimeOut	The SM shall set the maximum trap generation rate for all nodes in the subnet by initializing the *PortInfo.SubnetTimeOut* component in all ports as described in section 13.4.6.2.1 PortInfo:SubnetTimeout on page 613.
PortInfo:MasterSMLID	The SM shall store the LID of the port where it resides in the *PortInfo:MasterSMLID* of each port on the subnet.
PortInfo:MasterSMSL	The SM shall store the SL required for sending a non-SMP message to the SM using that LID in the *PortInfo:MasterSMSL* of each port on the subnet
PortInfo:PortPhysicalState	The default state on power-on is *polling* as described in Volume 2.
PortInfo:LinkDownDefaultState	The default state on power-on is *polling* as described in Volume 2.

Table 143 Initialization

Component	Description
PortInfo:VLHighLimit	The SM shall set the *Limit of High-Priority* limit for the number of bytes of high-priority packets that can be transmitted if the ports on both ends of a link may be operated with multiple data VLs as described in section 7.6 Virtual Lanes Mechanisms on page 153
PortInfo:M_Key	The SM may initialize the *PortInfo:M_Key* for each port on the subnet as described in section 16.2.3.1 ClassPortInfo on page 788. The rules for assigning these values is outside the scope of the specification.
PortInfo:M_KeyProtectBits	The SM may initialize the *PortInfo:M_KeyProtectBits* for each port on the subnet as described in section 14.2.4 Management Key on page 654. The rules for assigning these values is outside the scope of the specification.
PortInfo:M_KeyLeasePeriod	The SM may initialize the *PortInfo:M_KeyLeasePeriod* for each port on the subnet as described in section 14.2.4 Management Key on page 654. The rules for assigning these values is outside the scope of the specification.
PortInfo:M_KeyViolations	The SM shall clear the *PortInfo:M_KeyViolations* component for all ports on the subnet.
PortInfo:P_KeyViolations	The SM shall clear the *PortInfo:P_KeyViolations* component for all ports on the subnet.
PortInfo:Q_KeyViolations	The SM shall clear the *PortInfo:Q_KeyViolations* component for all ports on the subnet.
PortInfo:VLStallCount	The SM shall set a value for the *PortInfo:VLStallCount* as described in section 16.2.3.1 ClassPortInfo on page 788. The rules for assigning these values is outside the scope of the specification.
PortInfo:HOQLife	The SM shall set a value for the *PortInfo:HOQLife* as described in section 16.3.3.1 ClassPortInfo on page 798. The rules for assigning these values is outside the scope of the specification.
PortInfo:DiagCode	The SM may check the *PortInfo:DiagCode* of every port on the subnet. The rules for correcting faults detected on ports is outside the scope of the specification.
GUIDInfo	The SM may assign a GUID to ports to form GIDs as described in section 4.1.1 GID Usage and Properties on page 117. There is one default GID for each port. The requirements for setting additional GIDs is beyond the scope of the specification.
SwitchInfo:LinearFDBTop	On a switch that supports a linear forwarding table, the SM will program the highest LID to port mapping used as described in section 14.2.5.4 SwitchInfo on page 663.
SwitchInfo:DefaultPort	On a switch the support a random forwarding table, the SM must set the default port as described in section 18.2.4.3.2 Random Forwarding Table Requirements on page 855. The rules for assigning these values is outside the scope of the specification.

Table 143 Initialization

Component	Description
SwitchInfo:DefaultMulticastPrimaryPort	On a switch that support multicast, the SM shall set the *DefaultMulticastPrimaryPort* as described in section 18.2.4.3.3 Required Multicast Relay on page 856. The rules for assigning these values is outside the scope of the specification.
SwitchInfo:DefaultMulticastNotPrimaryPort	On a switch that support multicast, the SM shall set the *DefaultMulticastNotPrimaryPort* as described in section 18.2.4.3.3 Required Multicast Relay on page 856. The rules for assigning these values is outside the scope of the specification.
SwitchInfo:LifeTimeValue	The SM shall set a value for the *LifeTimeValue* as described in section 18.2.4.4 Transmitter Queuing on page 858. The rules for assigning these values is outside the scope of the specification.
VLArbitrationTable	VL arbitration described in section 7.6.9 VL Arbitration and Prioritization on page 161 shall be set by the SM for the output link of each CA, switch, and router.
SLtoVLmappingTable	The application of VL is described in section 7.6.6 VL Mapping Within a Subnet on page 159. The SM will initialize the SL-to-VL mapping tables. The rules for assigning these values is outside the scope of the specification. For switches, the SM checks for the existence of the *SLtoVLmappingTable* and initializes it if present.
P_KeyTable	The SM may initialize the P_Key table by setting entries in the *P_KeyTable* attribute for ports It may also enable P_Key checking in switches. The policy for assigning P_Keys is in general outside the scope of the specification. However, the SM shall ensure that one of the *P_KeyTable* entries in every node contains either the value 0xFFFF (the default P_Key, full membership) or the value 0x7FFF (the default P_Key, partial membership). The purpose of this specific P_Key value is to provide communication with Subnet Administration (see 15.4.2 Locating Subnet Administration on page 739).
LinearForwardingTable	Unicast forwarding tables will be set by the SM based on route policy decisions and Switch capabilities. The SM shall setup LID-to-port mappings if the Switch supports a Linear forwarding table as indicated by the *SwitchInfo:LinearFDBCap* component.
RandomForwardingTable	The SM shall setup LID/LMC range to port mappings based on route policy decisions and Switch capabilities. if the Switch supports a Random forwarding table as indicated by the *SwitchInfo:RandomFDBCap* component.
MulticastForwardingTable	The SM may setup LID to multi-port mappings based on route policy decisions and Switch capabilities. if the Switch supports a Multicast forwarding table as indicated by the *SwitchInfo:MulticastFDBCap* component.

14.4.4 PORT STATE TRANSITIONS

When power is applied to a device, its ports attempt to reach an operational state according to the steps described in Volume 2, Chapter 5 and section 6.2 Services provided by the Physical Layer. on page 137. A physical subnet is established when a group of devices are connected together and the state of a set of ports reaches operational state.

C14-63: A SM shall determine that a subnet is operational when the *PortInfo:Portstate* on the port where it resides is at the *initialize* state.

The SM may access the management entities of remote CAs, switches, and routers while the ports along the physical links are in *initialize* state since the SMI on that port will recognize a packet on QP0 and VL15, with a LID destination address 0xFFFF as referring to the SMA.

The SM may change the state of a port to *active*, *armed*, *initialize* or *down* that are described in section 14.4.4 Port State Transitions on page 697.

The SM may perform most port and device configuration activities while the *PortInfo:Portstate* is in the *initialize* state. However, all control and configuration options are also available in the *armed* state and the *active* state. In addition to the link level behaviors, the *PortInfo:Portstate* has an additional role cause it is manipulated by the SM to communicate to endnodes the readiness of the subnet. CAs and Routers may start sending packets on the subnet if one of its ports enters the *active* state. As a result, moving a port from *active* state is likely to be disruptive to subnet activity.

An SM that becomes the master SM may enable transmission of packets through the subnet at any time. This is accomplished after establishing routes by setting the switch forwarding tables and initializing the other attributes as described in section 14.4.3 Initialization Actions on page 694 for CAs, switches, and routers along those routes, and then setting the *PortInfo:Portstate* to *armed* for the ports along those routes. The SM changes the state of an Endnode from *armed* to *active* to signal to the Endnode that it may begin to send packets. Ports on switches and along that route and endnodes that are destinations of those packets will transition from *armed* to *active* automatically as described in section 14.4.4 Port State Transitions on page 697.

The SM may reset port related state by:

1) setting the *PortInfo:LinkDownDefaultState* is set to *polling*

2) setting the *PortInfo:Portstate* to the *down* state.

The *PortInfo:Portstate* should return to the *initialize* state after clearing its state as described by the link state machine in Figure 50 Link State Machine on page 144.

14.4.5 Subnet Sweeping

C14-64: After the subnet is up and running, the SM shall periodically gather information about topology changes, *PortInfo:CapabilityMask* changes, and *Notices* reported by nodes.

This is referred to as sweeping the subnet. The frequency of subnet sweeps is undefined for this architecture, as it will vary due to topology and other implementation considerations.

The SM detects topology changes by examining the port state of nodes in the subnet. For example, when the value of the *PortInfo.Portstate* component of a port changes from *down* to *initialize*, the SM will use directed routed packets to probe the other end of the link on that port to determine what has been added to the subnet. Conversely, if the *PortInfo.Portstate* component changes from *active* to *down*, the SM may perform operations such as updating switch forwarding tables to delete routes to the end-node(s) that are no longer accessible. To speed up detection of port state changes, switches support a *SwitchInfo:PortStateChange* component, described in Table 123 SwitchInfo on page 663, that the SM may examine. If the state of this component indicates that the state of one of the switch ports has changed, the SM may proceed to check the status of each port on that switch.

14.4.6 Authentication

During initialization of a SMP, the SM may fill in the *MADHeader:M_Key* field of the SMP with the value that matches the M_Key stored in the destination port if it expects the destination management entity to check it.

C14-65: The SM shall not check the *MADHeader:M_Key* stored in a *SubnGetResp(*)*.

C14-66: If the SM receives a validated SMP containing a *SubnSet(SMInfo)* or *SubnGet(SMInfo)* and the *PortInfo:M_Key* component is zero, then the SM shall generate a *SubnGetResp*.

C14-67: If the SM receives a validated SMP containing a *SubnSet(SMInfo)* or *SubnGet(SMInfo),* and the *PortInfo:M_Key* component is non-zero, and M_Key matching, if required, is successful according to the rules specified in section 14.2.4 Management Key on page 654, then the SM shall generate a *SubnGetResp*. Otherwise the *SubnSet(SMInfo)* or *SubnGet(SMInfo)* is silently discarded.

C14-68: When a Master SM receives a SMP containing a *SubnTrap()*, it shall not check that the *MADHeader:M_Key* field matches the *PortInfo:M_Key* of the port where the SMP was received.

The *SMInfo:SM_Key* is used by the Master SM to authenticate other standby SMs and master SMs, in the case of a subnet merge, on the subnet. Exactly how the key is used is implementation specific. A SM should fill the SM_Key in the *SMInfo:SM_Key* component in a response if it expects the requesting SM to check it.

14.4.7 SM Disable Mechanism

C14-69: If a SM can reside on a port, a vendor defined, out-of-band mechanism shall be provided that when asserted will disable the capability of running a SM from that port and the state of the mechanism shall be indicated in the *Portinfo:CapabilityMask.IsSMdisabled* bit.

C14-70: When the *Portinfo:CapabilityMask.IsSMdisabled* bit is asserted, the port behavior shall be:

- *SubnSet(SMInfo)* or *SubnGet(SMInfo)* sent to that port shall be discarded
- *SubnSet(*)* or *SubnGet(*)* shall not be sent from that port
- The *Portinfo:CapabilityMask.IsSM* bit for that port shall not be set.

C14-71: When *Portinfo:CapabilityMask.IsSMdisabled* is not-asserted, the port behavior shall be:

- *SubnSet(SMInfo)* or *SubnGet(SMInfo)* sent to that port will be forwarded to management entities if the appropriate entity is operational
- *SubnSet(*)* or *SubnGet(*)* may be sent by management entities from the port
- The *Portinfo:CapabilityMask.IsSM* bit is controlled by management entities behind that port

The mechanism for changing the state of the *Portinfo:CapabilityMask.IsSMdisabled* bit is beyond the scope of the specification.

C14-72: The state of the *Portinfo:CapabilityMask.IsSMdisabled* on a port shall be changeable at any time while the port is operational.

Changing the state of *Portinfo:CapabilityMask.IsSMdisabled* from asserted to not-asserted while the port is otherwise operational may cause a SM to startup, but that is beyond the scope of the specification.

CHAPTER 15: SUBNET ADMINISTRATION

15.1 INTRODUCTION AND OVERVIEW

This chapter defines IBA Subnet Administration (SA) communication and the function of that communication: the MADs used, and the functions they are associated with.

C15-1: Every IBA subnet must provide an SA.

15.1.1 SA FUNCTION

Through the use of Subnet Administration class MADs, SA provides access to and storage of information of several types, some optional.

C15-2: The information that shall be provided by SA is specified in Table 148 Subnet Administration Attributes (Summary) on page 710.

The types of information involved are:

- Information that endnodes require for operation in a subnet. Such information includes paths between endnodes, notification of events, service records, etc. This information is required.
- information that is non-algorithmic, typically. Information that cannot be recovered algorithmically by inspection of the network after a power-on or initialization event. Such information includes partitioning data, M_Keys, SL to VL mappings, etc. This information is required to allow off-line migration from one vendor's subnet management implementation to another's. This is required.
- Information that may be useful to other management entities such as standby SMs, who may, for example, wish to use it to maintain synchronization with the master SM. Such information includes subnet topology data, switch forwarding tables, etc. This is optional.

In order to perform these functions, SA includes three functions:

- A reliable multipacket data transfer protocol, required because the information sent to or retrieved by SA in many cases is larger than will fit into a single MAD unreliable datagram
- A query subsystem required to identify the information to be sent and received
- An event-forwarding subsystem that forwards SM-received traps and notices to subscribed parties.

The actual SA implementation is outside the scope of architecture. The actual access QP and DLID may be redirected by the GSI.

15.1.2 RELATIONSHIP BETWEEN SA AND THE SM

Much, but not all, of the information provided by SA is created or collected by the SM. SA must therefore have a close relationship with the master SM. That relationship is defined as follows:

- SA is part of the SM. Its functions are discussed separately from the SM only for convenience of description. This descriptive convenience is not intended to imply or require any particular implementation organization of the SM (or SA) by any vendor.
- As is the case for any class of IB management, SA functions may be implemented on a node separate from the one holding the SM; whether this is done is vendor-specific. If any SM function is implemented at a location different from the one identified as holding the SM, including but not limited to SA functions, any or all communication between that function and any other SM elements is vendor-specific.

C15-3: Should an SM be elected master SM, all its components must also be implicitly elected master, including but not limited to SA, however they may be implemented. If an SM ceases to be master, all of its components, including but not limited to SA, must cease responding to messages from client nodes.

15.1.3 OVERVIEW

The remainder of this chapter first defines the MADs used by SA. It also defines the reliable multipacket transport protocol; and then the operation of SA. The SA operations described include: locating the SA, the SA methods and their operation. Also described are identification of information records, access restrictions that must be implemented, versioning, and event forwarding.

15.2 SA MADS

This section defines the MADs sent and received by SA.

C15-4: The SA MADs must use the Generic Services Interface (GSI), and adhere to all GSI rules of use. Like all MADs, they must conform to MAD use as specified in 13.4 Management Datagrams on page 603.

15.2.1 SA MAD FORMAT

Table 144 on page 703 shows the Subnet Administration datagram format that will be used, and Table 145 Subnet Administration Fields on page 703 defines the fields contained in the Subnet Administration datagram.

Table 144 Subnet Administration Format

bytes				
0-24	Standard MAD Header (see Figure 143 on page 605)			
24	SA_KEY			
28				
32	SM_KEY			
36				
40	Segment Number			
44	Payload Length			
48	Fragment Flag	Edit Modifier	Window	
52	EndRID			
56	ComponentMask			
60				
64	Admin Data (192 bytes)			
...				
252				

Table 145 Subnet Administration Fields

Field	Length	Description
SA_KEY	64 bits	Subnet Administration key value If 0 no prior admin queries performed. Ignored by non-query methods.
SM_KEY	64 bits	Subnet Manager verification key. Refer to Chapter 14.
Segment Number	32 bits	Segment number of a segmented Subnet Admin packet.
Payload Length	32 bits	The number of valid data bytes in data stream, if a multi-packet data sequence in the first packet of a request or a response. 15.3.1.2 Payload Length on page 728.
FragmentFlag	8 bits	refer to 15.3.1.3 Fragment Flag on page 728

Table 145 Subnet Administration Fields

Edit Modifier	8 bits	An enumeration with the following values: 0 - Add record modifier 1 - Delete record modifier 2 - Edit record modifier 3 to 255 - Reserved
Window	16 bits	For multipacket operations. Refer to Windowing 15.3.1.4
End RID	32 bits	Used exclusively for indicating the ending Record identifier (RID) for table query and manipulation operations. Set to zeroes for all other operations.
ComponentMask	64 bits	Used to indicate attribute components to be used for query and edit operations. Bit 0 maps to first attribute, bit 1 the second attribute, and so forth. A bit set to one indicates attribute component is used to form query or edit operation, otherwise field is to be ignored.
Subnet Admin Data	1536 bits/ 192 bytes	Data field where attribute content is stored.

15.2.1.1 SA-SPECIFIC CLASSPORTINFO:CAPABILITYMASK BITS

The Subnet Administration class uses two class-specific bit of the ClassPortInfo:CapabilityMask:

- bit 8 is defined and named “IsSubnetOptionalRecordsSupported”
- bit 9 is defined and named “IsUDMulticastSupported.”

C15-5: If IsSubnetOptionalRecordsSupported=1, SA must support all records listed as optional in Table 148 Subnet Administration Attributes (Summary) on page 710 except for MCGroupRecord and MCMemberRecord, and all the methods listed as optional in Table 146 Subnet Administration Methods on page 705. This bit must not be used to indicate support for only some of those records and methods. If IsSubnetOptionalRecordsSupported=0, SA does not support those records and methods.

C15-6: If IsUDMulticastSupported=1, SA must support MCGroupRecord and MCMemberRecord as listed in Table 148 Subnet Administration Attributes (Summary) on page 710. If IsUDMulticastSupported=0, SA does not support those records.

See 13.4.8.1 ClassPortInfo on page 619 for a description of the ClassPortInfo:CapabilityMask.

15.2.2 SUMMARY OF METHODS

Table 146 Subnet Administration Methods on page 705 summarizes the methods provided by the Subnet Administration class. Several of these are common methods described in 13.4.5 Management Class Methods on page 607; some are unique to this class. Subnet Administration methods are described in more detail in 15.4 Operations on page 737.

Those methods which use the Multipacket Reliable Transport Protocol (see section 15.3 on page 727) have "Yes" in the "Multi-Packet" column.

C15-7: SA must support all the methods listed as required in Table 146 on page 705.

o15-1: SA may support the methods listed as optional in that table; either all such methods must be supported, or none.

Table 146 Subnet Administration Methods

Method Type	Value	Multi-Packet?	Optional or Required	Description
SubnAdmGet()	0x01	No	Required	Request a get (read) of an attribute from a node.
SubnAdmGetResp()	0x81	No	Required	The response from an attribute get or set request.
SubnAdmSet()	0x02	No	Required	Request a set (write) of an attribute in a node. The object will issue a SubnAdmGetResp() as a response.
SubnAdmInform()	0x10	No	Required	Request an event subscription.
SubnAdmInformResp()	0x90	No	Required	Reply to an event subscription request.
SubnAdmReport()	0x06	No	Required	Forward an event previously subscribed for.
SubnAdmReportResp()	0x86	No	Required	Reply to a SubnetAdmReport method.
SubnAdmGetTable()	0x12	No	Required	Subnet Manager table request.
SubnAdmGetTableResp()	0x92	Yes	Required	Subnet Manager table request response.
SubnAdmGetBulk()	0x13	No	Optional	Dump Subnet Manager data request.
SubnAdmGetBulkResp()	0x93	Yes	Optional	Dump Subnet Manager data response
SubnAdmConfig()	0x15	Yes	Required	Request to configure
SubnAdmConfigResp()	0x95	Yes	Required	Response to configuration request

15.2.3 SUBNET ADMINISTRATION STATUS VALUES

Table 147 Administration MAD Status Field Bit Values

Name	Bit	Meaning
Common bit values	0-7	See 13.4.7 Status Field on page 617
ERR_KEY_STALE	8	Supplied key is stale, need to reload table records/get new lease.
ERR_REQ_INVALID	9	Supplied request or update is invalid.
	10-11	Reserved

Table 147 Administration MAD Status Field Bit Values

Name	Bit	Meaning
ERR_REFUSED	12	Policy violation
RESERVED2	13-15	Reserved for future use.

15.2.4 ATTRIBUTES AND RECORD ATTRIBUTES

The contents of the Admin Data field of SA MADs are, in effect, *data records* of various types. Since the attribute field of the standard header identifies Admin Data formats and semantics, the term Record Attribute (RA) is used to refer to these attribute types.

RAs may be spoken of as *logically stored* in SA; while some must actually be stored, whether others are actually stored or are computed in response to a query is implementation-dependent.

Three ways exist for RAs to be stored by SA:

- Attributes are captured and deposited by the Master SM in the course of routing and sweeping the subnet.
- Attributes are stored as a result of traps that are captured and logged by the Master SM.
- Attributes are logged by administrative software updating subnet configuration data. Some of these may, ultimately, have been entered by a customer; for example, partition records recording customer's partitioning configuration input.

15.2.4.1 RECORD ATTRIBUTES (RA)

To distinguish them, RAs use a naming convention: They are all called XXXRecord, where XXX is the name by which the data is known. When the data is an SM attribute the XXX is the SM attribute name (see 14.2.5 Attributes on page 658). For example, the NodeInfo subnet management attribute is reflected in the RA NodeRecord. Other RAs have no counterpart in SM attributes, but use this convention anyway; PathRecords and MCGroupRecords are examples.

Every RA has identifiers associated with it called Record Identifiers (RIDs), using the layout illustrated in Figure 165 on page 707. Each RA always has its own RID (as illustrated), and in some cases there is also a

RID that indicates relations to other RAs (not illustrated). RIDs are defined in 15.2.4.3 on page 708; they serve several purposes:

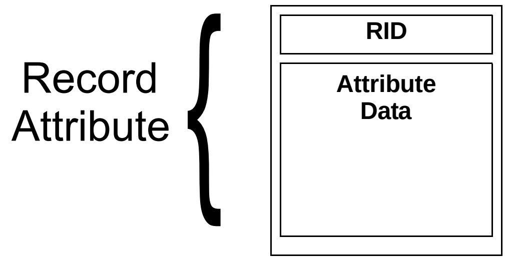


Figure 165 Record Attribute

- They allow query and modification of the attribute itself
- They allow local relational organization of obtained data, if desired (this is implementation dependent)
- They allow further queries of SA using these RIDs

As a response to a query method, the RAs are organized into *tables* when multiple RAs are returned.

15.2.4.1.1 STATE RECORD RAS

State records are RAs available for query, exclusively. They cannot be edited. These records reflect dynamic topology data, constantly being used and manipulated by the SM. Examples of these kinds of RAs are the PortInfoRecord and NodeRecord.

15.2.4.1.2 CONFIGURATION RECORD RAS

Configuration records RAs are not normally updated by the SA. ServiceRecord (a type of configuration record RA; see Section 15.2.5.15 on page 716) is an exception which is deleted when the service lease expires. Examples of Configuration records are the ServiceRecord and the InformRecord. These RAs are subject to edit by management applications by the administrative interface.

A change in a configuration record implies possible activity by the SM to bring a subnet into compliance with the new RA. In some cases this is unnecessary; for example, changes to ServiceRecords imply no change other than to provide these new RAs on later queries. Other changes to configuration records could have more dramatic effects. For example, changes to the RangeRecords may cause the master SM to update range records on standby SMs to bring the standbys into compliance with the master SM (this is implementation dependent).

15.2.4.2 RECORD TABLES AND BULK RECORDS

Collections of RAs transferred by communications with subnet administration are called tables. These tables consist of a single type of RA, such as NodeRecords.

Collections of all the tables together in a single collection is referred to as bulk.

15.2.4.3 RECORD IDENTIFIERS (RIDS)

All RAs have at least a single RID. This RID is used to provide a RA with a specific unique identifier for query and edit operations. Most RAs have a single RID, referred to as a major RID, which is the following format:

LID	PortNumber	Enumeration
16 bits	8 bits	8 bits

Figure 166 General RID model

The LID will be the base LID of the port in question.

For switches the PortNumber is also used to specify to which port a RA is related to. This is not used for channel adapters, and set to 0.

The enumeration is used where more than one, but less than 256 RAs are related to a specific port.

For example, several GUIDInfo RAs for CA port with a LID of 5 will use the LID portion of the RID to refer to the CA ports base LID, set the PortNumber portion of the RID to 0 (as it is unused for CAs), and will use the enumeration of 1 to refer to the 1st GuidInfoRecord RA related to LID 5, and enumeration value of 2 to refer to the second GuidInfoRecord RA related to LID 5, and so on.

RAs that cannot be cataloged with the general RID model are instead identified with the unique RID, which is as follows:

	Unique RID(32 bits)		
Related NodeRecord RID {	LID	PortNumber	Enumeration

Figure 167 Unique RID model

The Unique RID, another type of major RID, is any unique value for a RA coupled with a secondary RID relating the RA to a specific NodeRecord.

In this way all RAs can be related back to a given port that have a determined relation. This second RID is referred to as the related RID.

15.2.4.3.1 SPECIAL RID USAGE

SlToVlMappingTableRecord uses the General RID, using the PortNumber to indicate the input port and the enumeration to indicate the output port this record references.

VlArbitrationTableRecord the General RID is used, where the Portnumber corresponds to the Portnumber of the switch this attribute is assigned to, and the enumeration is used to specify the part of the table assigned, as defined in Chapter 14.

LinearForwardingTableRecords use the General RID model, however, they use the PortNumber and Enumeration fields as a single16 bit integer field instead of as two separate fields to enumerate up to 767 possible records for a switch.

RandomForwardingTableRecords use the use the General RID model, however, they use the PortNumber and Enumeration fields as a single16 bit integer field instead of as two separate fields to enumerate up to 3071 possible records for a switch.

MulticastForwardingTableRecords uses the General RID model, but again the PortNumber and Enumeration fields are viewed as a single value to specify the 14 meaningful bits of the MulticastForwardingTable assigned to a switch, the two low order bits are not used and set to 0.

PartitionRecords uses the General RID to refer to the CA, Switch or Router this attribute is assigned to, will also use the PortNumber to specify the Port this record is assigned to. The Unique RID is a unique value assigned by SA to the P_KeyTableRecord.

15.2.4.4 LID ALIASING

LID assignment can change in the subnet due to a variety of conditions. Ports that have multiple LIDs assigned to them by use of the LMC can be addressed by the base LID, or by any LID value that could be valid for that RA by exercise of the LMC. The ability to use any of the values of the LID supported for an object by use of the LMC is called LID aliasing.

15.2.5 ATTRIBUTES

This section first provides a summary of all the SA attributes, and then lists the data format of each, with descriptions where warranted.

15.2.5.1 SUMMARY OF ATTRIBUTES

C15-8: SA must process all the attributes listed as required in Table 148 on page 710, which summarizes the SA attributes.

o15-2: SA may also process the attributes listed as optional in Table 148 on page 710. These form two groups, one for UD Multicast, and the other for other optional records. Either all of a group must be processed, or none of that group. See 15.2.1.1 SA-Specific ClassPortInfo:CapabilityMask Bits on page 704 for the definition of those groups.

RAs are noted in that table, along with the Subnet Management attribute they contain; see also 15.2.1.1 SA-Specific ClassPortInfo:CapabilityMask Bits on page 704.

Configuration and State records are noted in Table 148, along with the Subnet Management attribute they contain.

Table 148 Subnet Administration Attributes (Summary)

Attribute Name	Container for (if applicable)	Attribute ID	Optional / Required	Record Attributes	Configuration	State	Description
ClassPortInfo	N/A	0x0001	R	n	n	n	Class information; see 13.4.4
Notice	N/A	0x0002	R	n	n	n	Notice information; see 13.4.8.2
InformInfo	N/A	0x0003	R	n	n	n	Subscription (Inform) Information; see 13.4.8.3
NodeRecord	NodeInfo & NodeDescription	0x0011	R	Y	n	S	NodeInfo record
PortInfoRecord	PortInfo	0x0012	R	Y	n	S	PortInfo record
SltoVlMapping-TableRecord	SltoVlMappingTable	0x0013	R	Y	n	S	SltoVlMappingTable record
SwitchRecord	SwitchInfo	0x0014	O	Y	n	S	SwitchInfo record
LinearForwarding-TableRecord	LinearForwardingTable	0x0015	O	Y	n	S	Linear Forwarding database entry records
RandomForwarding-TableRecord	RandomForwardingTable	0x0016	O	Y	n	S	Random forwarding database entry records
MulticastForwarding-TableRecord	MulticastForwardingTable	0x0017	O	Y	n	S	Multicast Forwarding database entry records
SMInfoRecord	SmInfo	0x0018	O	Y	n	S	SmInfo record
InformRecord	InformInfo	0x00F3	O	Y	C	S	InformInfo record
NoticeRecord	Notice	0x00F4	O	Y	n	S	Notice or trap record
LinkRecord	N/A	0x0020	O	Y	n	S	Link record

Table 148 Subnet Administration Attributes (Summary)

Attribute Name	Container for (if applicable)	Attribute ID	Optional / Required	Record Attributes	Configuration	State	Description
GuidInfoRecord	GUIDInfo	0x0030	O	Y	n	S	GIDS assigned to a port
ServiceRecord	N/A	0x0031	R	Y	C	S	Service advertisement record
PartitionRecord	PartitionInfo	0x0033	R	Y	n	S	Partition records
RangeRecord	N/A	0x0034	R	Y	C	S	Range records
PathRecord	N/A	0x0035	R	Y	n	S	Subnet path information
VLArbitrationRecord	VLArbitrationTable	0x0036	R	Y	n	S	VL arbitration record
MCGroupRecord	N/A	0x0037	O	Y	C	S	multicast group records
MCMemberRecord	N/A	0x0038	O	Y	C	S	multicast member record
SAResponse	N/A	0x8001	O	n	n	n	Container for subnet query response

Table 149 Subnet Administration Attribute / Method Map on page 711 associates SA attributes with methods.

SA must allow use of all the specified methods with each attribute.

Table 149 Subnet Administration Attribute / Method Map

Attribute	Get	Set	Inform	Report	GetTable	GetBulk
ClassPortInfo	x					
Notice				x		
InformInfo			x			
NodeRecord	x				x	x
PortInfoRecord	x				x	x
SltoVlMappingTableRecord	x				x	x
SwitchRecord	x				x	x
LinearForwarding-TableRecord					x	x
RandomForwarding-TableRecord					x	x

Table 149 Subnet Administration Attribute / Method Map

Attribute	Get	Set	Inform	Report	GetTable	GetBulk
MulticastForwarding-TableRecord					x	x
VLArbitrationRecord					x	x
SMInfoRecord	x				x	x
InformRecord	x	x			x	x
NoticeRecord	x				x	x
LinkRecord	x				x	x
GUIDInfo Record	x				x	x
ServiceRecord	x	x			x	x
PartitionRecord	x				x	x
RangeRecord	x	x			x	x
PathtRecord	x				x	
MCGroupRecord	x	x			x	x
MCMemberRecord	x	x			x	x
SAResponse						x

The detailed layouts of all the SA-specific records follows. RAs which are containers for Subnet Management records require little description beyond their layout; others have more extensive descriptions

15.2.5.2 NODERECORD

Table 150 NodeRecord

Component	Length(bits)	Offset(bits)	Description
NodeRID	32	0	The node's RID, used as an RID record
NodeInfo	320	32	NodeInfo Record contents
NodeDescrip-tion	512	352	NodeDescription Record contents

If a node has multiple ports on the same subnet, there will be multiple NodeRecords available for that node from the SA, one for each possible PortGUID value of NodeInfo for that node.

15.2.5.3 PORTINFORECORD

Table 151 PortInfoRecord

Component	Length(bits)	Offset(bits)	Description
NodeRID	32	0	RID of this record
PortInfo	432	32	PortInfo Attributes record

15.2.5.4 SLTOVLINFORECORD

Table 152 SltoVLMappingTableRecord

Component	Length(bits)	Offset(bits)	Description
SlVLRID	32	0	Unique RID of this record
NodeRID	32	32	Node RID this record is referencing
SlVLMapping	64	64	SlToVLMapping attribute

15.2.5.5 SWITCHRECORD

Table 153 SwitchRecord

Component	Length(bits)	Offset(bits)	Description
NodeRID	32	0	RID of this record
SwitchInfo	132	32	Contents of SwitchInfo Attribute

15.2.5.6 LINEARFDBRECORD

Table 154 LinearFdbRecord

Component	Length(bits)	Offset(bits)	Description
LinearFdbRID	32	0	RID of this forwarding record
NodeRID	32	32	Reference to related node
LinearFdbInfo	512	64	Contents of Linear forwarding DB

15.2.5.7 RANDOMFDBRECORD

Table 155 RandomFdbRecord

Component	Length(bits)	Offset(bits)	Description
RandomFd-bRID	32	0	RID of this forwarding record
NodeRID	32	32	RID reference to node
RandomFdb	512	64	Contents of Random Forwarding Table

15.2.5.8 MULTICASTFORWARDINGRECORD

Table 156 MulticastForwardingRecord

Component	Length(bits)	Offset(bits)	Description
McastRID	32	0	RID of this forwarding record
NodeRID	32	32	RID reference to node.
MulticastFor-wardingTable	512	64	Contents of Multicast Forwarding Table

15.2.5.9 VLARBITRATIONRECORD

Table 157 VLArbitrationRecord

Component	Length(bits)	Offset(bits)	Description
VLArbRID	32	0	RID of this forwarding record
NodeRID	32	32	RID reference to node.
VLArbitration	512	64	Contents of VLArbitration Attribute

15.2.5.10 SMINFORECORD

Table 158 SMInfoRecord

Component	Length(bits)	Offset(bits)	Description
NodeRID	32	0	RID of related NodeInfo record
SMInfo	168	32	Contents of SMInfo Attributes of given SM

15.2.5.11 PARTITIONRECORD

Table 159 PartitionRecord

Component	Length(bits)	Offset(bits)	Description
PartitionRID	32	0	Unique RID of this record
NodeRID	32	32	RID of related NodeInfo record
P_KeyTable	512	64	Contents of P_KeyTable attribute

15.2.5.12 INFORMRECORD

Table 160 InformRecord

Component	Length(bits)	Offset(bits)	Description
InformRID	32	0	RID of this record
NodeRID	32	32	RID of related NodeInfo record
Inform	344	64	Content of Inform attribute record

15.2.5.13 NOTICERECORD

Table 161 NoticeRecord

Component	Length(bits)	Offset(bits)	Description
NoticeRID	32	0	RID of this notice/trap record.
NodeRID	32	32	RID of related NodeInfo record
Notice	512	64	Content of Notice attribute record

15.2.5.14 LINKRECORD

Table 162 LinkRecord

Component	Length(bits)	Offset(bits)	Description
LinkRID	32	0	RID of this record
FromLID	16	32	From InfiniBand address
ToLID	16	48	To InfiniBand address
FromPort	8	64	From port number
ToPort	8	76	To port number

Link records are synthesized by the subnet administration to serve as informational topology data for management entities in need of such data.

15.2.5.15 SERVICERECORD

Table 163 ServiceRecord

Component	Length(bits)	Offset(bits)	Description
ServiceRID	32	0	RID of this service record
ServiceLease	32	32	Lease period remaining for this service, in seconds. 0xFFFFFFFF is an indefinite lease.
Partition	16	64	Partition of this Service
ServiceSpecificFlags	12	80	Information related to this service. Content is service specific.
ServiceGenericFlags	4	92	Generic information related to this service. The interpretation of individual bits in the field is as follows: Bit 0: Indirection, set if the service provider may redirect requests; Bit 1: DHCP-capable, set if a DHCP server or a Directory Agent may automatically register this service by querying the SA; Bit 2: Reserved. Bit 3: Reserved.
ServiceName	992	96	UTF-8 encoded, null-terminated name of the service.
ServiceGID	320	1088	Text representation (compliant with IPv6 conventions[a]) of the port GID for the service, null-terminated
ServiceID	128	1408	String of 16 hexadecimal digits, including any leading zeros, not null-terminated

a. Hinden, R. and Deering, S., *RFC 2373: IP Version 6 Addressing Architecture*, July 1998 (Section 2.2 of the document describes rules for text representation of IPv6 addresses.)

Service records serve the purpose of first level or “bootstrap” advertisement of basic services that cannot be found prior to query of the SA. These could be services such as boot services, or name or directory services.

ServiceRecords are not intended to do more than to provide a first level directory to other applications and services normally associated with a network. If there are more than one ServiceLocations associated with a ServiceName, there are multiple Service Records; one for each ServiceLocation.

15.2.5.15.1 SERVICENAME

ServiceRecord:ServiceName is a character string identifying what service is being sought (for example “tftp”, “CFM.IBTA”, “sendmail”, and so on). This is a 124-byte long, UTF-8 encoded, null-terminated string.

15.2.5.16 RANGERECORD

Table 164 RangeRecord

Component	Length(bits)	Offset(bits)	Description
RangeRID	32	0	RID of the range record
NodeRID	32	32	RID of related NodeInfo record
FmAssigned	64	64	GUID of SM allotted range
FromRange	16	128	Value of beginning of range
ToRange	16	144	Value of end of range

Range records specify ranges of LIDs. They exist to allow avoidance of LID conflicts in some cases. A Master SM can use them to provide ranges of LIDS to standby SMs, thereby enabling the standby SMs to use known unique ranges if a subnet they control is independently initialized. SubnAdmConfig() can be used to “push” these from master to standby, and Get operations can be used by standbys to get ranges from the master.

15.2.5.17 PATHRECORD

Table 165 PathRecord

Component	Length(bits)	Offset(bits)	Required For GetTable Request	Description
PathRID	32	0		RID of this PathRecord
Reserved0	32	32		Offset for alignment
DGID	128	64		Destination GID to establish path to
SGID	128	192	X	Source GID to establish path from
DLID	16	320		Destination LID
SLID	16	336		Source LID
RawTraffic	1	352		Raw Packet path 0 - IB Packet (P_Key must be valid) 1 - Raw Packet traffic (No P_Key)
Reserved3	3	353		Reserved (Ignored)

Table 165 PathRecord

Component	Length(bits)	Offset(bits)	Required For GetTable Request	Description
FlowLabel	20	356		FlowLabel (to be used in the GRH if GRH used)
HopLimit	8	376		Hop limit (to be used in the GRH if GRH used)
TClass	8	384		TClass (to be used in the GRH if GRH used)
Reserved1	1	392		Reserved (Ignored)
NumbPath	7	393	X	Maximum number of paths to return (or be returned) If more paths exist, the paths returned meet the requirements in the GetTable request, but are limited to this number of entries (implementation dependent)
P_Key	16	400		Partition Key for this path
SL	16	416		Service level - bit significant (MSB 16...LSB 0)
MtuSelector	2	432		0-greater than MTU specified 1-less than MTU specified 2-exactly the MTU specified
Mtu	6	434		Enumeration of the MTU required: 1: 256 2: 512 3: 1024 4: 2048 5: 4096 5-63: reserved
RateSelector	2	440		0-greater than rate specified 1-less than rate specified 2-exactly the rate specified
Rate	6	442		Enumeration of the rate: 1: 1 Gb/sec. 2: 2.5 Gb/sec. 3: 10 Gb/sec. 4: 30 Gb/sec. 5-64: reserved
PacketLife-TimeSelector	2	448		0-greater than PacketLifeTime specified 1-less than PacketLifeTime specified 2-exactly the PacketLifeTime specified
PacketLife-Time	6	450		Accumulated packet life time for the path specified by an enumeration derived from in units of 4.096 microseconds * 2^PacketLifeTime
Reserved2	56	456		Offset for alignment

The PathRecord RA is used to request routing information between endnodes. Its results are required to create connections and perform other tasks. The data returned in a PathRecord is usually generated, based on the routing algorithm used by the SM. SA may issue a secondary redirect to another service to respond to the request. Redirected requests are serviced using the same class requests and responses.

PathRecords can use the Administration Query Subsystem (15.4.5 on page 739) to request paths with desired properties. Using the component mask, the requester can build a GetTable request and supply the known fields in the record; the reply from SA will supply the response entries which match the request. For example, by setting the Component Mask used to cause everything but the SGID and DGID to be ignored, a *SubnAdmGetTable()* will return PathRecords for all paths from the SLID to the DLID. By selectively specifying the qualities desired, a path with any given qualities can be requested.

Normally the DGID is known (or at some point learned i.e. name service). But during a “boot” sequence, it may be useful to leave it unspecified, thus returning paths to all endnodes reachable from an SGID.

C15-9: SA shall provide a wildcard PathRecord query such that when the DLID and DGID are both specified as component mask entries of 0 in a query, and the SLID is that of the requester, that query shall return a single path record to each reachable port. Which path returned for each reachable port is indeterminate.

This can be used as the equivalent of a operating system’s “bus walk” that finds all reachable devices. Note that 15.4.1 Restrictions on Access on page 737 requires that the only paths returned are to devices which are visible under the partitioning arrangements in force.

Table 166 on page 719 is an example of the MAD header of a request for PathRecords (the data field is shown in the subsequent table):

Table 166 Example PathRecord Request MAD Header

Component	Value	Description
Header: Method	0x12	SubnAdmGetTable
Header: MADstatus	0	Set to zero (ignored)
Header:AttributeID	0x00035	Specifying the PathRecord
Header:Attribute Modifier	0xFFFFFFFF	Specifying a Query request matching the record in the data field1
Header: EndRID	0x0000	Set to 0, ignored by SA
TransactionID	0x11223344	Transaction ID (to be returned in the response)

Table 166 Example PathRecord Request MAD Header

Component	Value	Description
PayLoad Length	0x75	1 * sizeof (PathRecord) + Header=1*0x35+0x40

Table 167 Example PathRecord Request on page 720 shows what the data field in request would look like if up to two paths are requested to access a service. The path specifications being requested are:

- the MTU is no larger than 1024,
- the rate must be at 2.5 Gb/sec.,
- the path must be in the partition which the requester has access to,
- using service level 3,
- for any Packet Life Time cost,
- for any TClass,
- for raw or IB traffic,
- any flow label,
- or any number of “hops”

Table 167 Example PathRecord Request

Component	Component Mask Bit	Value	Meaning/Implication
PathRID	0	0	Query any record
Reserved0	0	0	Padding
DGID	1	Service global ID	GID of the service to communicate with (typically from a name service)
SGID	1	Local global ID	Source GID address
DLID	0	0	Indicates to the Subnet Administration that the path may be to any port on the destination node.
SLID	0	0	Indicates to the SA that the path may be from any port on the local node.
RawTraffic	1	0	IB traffic (P_Key will be valid)
Reserved3	0	0	ignored - set to 0
FlowLabel	0	0	FlowLabel (to be used in the GRH if GRH used)
HopLimit	0	0	Hop limit (to be used in the GRH is GRH used)
TClass	0	0	TClass (to be used in the GRH if GRH used)
Reserved1	0	0	ignored - set to 0

Table 167 Example PathRecord Request

Component	Component Mask Bit	Value	Meaning/Implication
NumbPath	1	2	Requesting only 2 paths which meet these specifications
P_Key	1	0x1234	P_Key to be used for this path)
SL	1	0x8	SL for this path (bit 3 means SL3)
MtuSelector	1	1	Paths must use an MTU less then 1024
Mtu	1	4	Mtu is less than 2048, i.e., 1024 or lower
RateSelector	1	2	Path rate must be exactly 2.5Gb/sec.
Rate	1	2	Rate is equal to 2.5Gb/sec.
PacketLifeTimeSelector	0	0	Return paths with any PacketLifeTime
PacketLifeTime	0	0	
Reserved2	0	0	Padding

For this example the following is a possible resulting response header and the PathRecords found in the data field of the response:

Table 168 Example PathRecord Response MAD Header

Component	Value	Meaning/Implication
Header: Method	0x92	SubnAdmGetTableResp
Header: MAD status	0	Good Status
Header:AttributeID	0x00035	Specifying the PathRecord
Header:Attribute Modifier	0x0000	Unused, set to 0
Header: EndRID	0x0000	Unused, set to 0
TransactionID	0x11223344	Same as in request
PayLoad Length	0xA7	2 * sizeof (PathRecord) + Header=2*0x35+0x40

The following is the records in the data field of the resulting response:

Table 169 Example PathRecord Response

Component	Value	Meaning/Implication
PathRID	0x01	RID of this Path record
Reserved0	0	

Table 169 Example PathRecord Response

Component	Value	Meaning/Implication
DGID	Service global ID	GID of the service to communicate with (typically from a name service)
SGID	Local global ID	Source GID address
DLID	0x0008	The LID assigned to the port where this service can be reached
SLID	0x000A	The LID assigned to the port where this service can be accessed for the SGID
RawTraffic	0	IB traffic (P_Key will be valid)
Reserved3	0	ignored - set to 0
FlowLabel	0	Default FlowLabel since this is an intra-subnet DGID
HopLimit	0	Default HopLimit since this is an intra-subnet DGID
TClass	0	Default TClass since this is an intra-subnet DGID
Reserved1	0	ignored - set to 0
NumbPath	2	2 paths are being returned
P_Key	0x1234	P_Key to be used for this path)
SL	0x8	This path has a SL of 3 (bit 3)
MtuSelector	2	Paths is exactly 1024
Mtu	3	
RateSelector	2	Path rate is exactly 2.5Gb
Rate	2	
PacketLifeTimeSelectort	2	Path PacketLifeTime is 4.096usec * 2^2=16.384 usec exactly.
PacketLifeTime	2	
Reserved2	0	
PathRID	0x03	RID of this record
Reserved0	0	
DGID	Service global ID	GID of the service to communicate with (typically from a name service)
SGID	Local global ID	Source GID address
DLID	0x0009	The LID assigned to the port where this service can be reached
SLID	0x000A	The LID assigned to the port where this service can be accessed for the SGID

Table 169 Example PathRecord Response

Component	Value	Meaning/Implication
RawTraffic	0	IB traffic (P_Key will be valid)
Reserved3	0	ignored - set to 0
FlowLabel	0	Default FlowLabel since this is an intra-subnet DGID
HopLimit	0	Default HopLimit since this is an intra-subnet DGID
TClass	0	Default TClass since this is an intra-subnet DGID
Reserved1	0	ignored - set to 0
NumbPath	2	2 paths are being returned
P_Key	0x1234	P_Key to be used for this path
SL	0x8	This path has a SL of 3 (bit 3)
MtuSelector	2	Path Mtu is exactly 512
Mtu	2	
RateSelector	2	Path rate is exactly 2.5Gb/sec.
Rate	2	
PacketLifeTimeSelector	2	Path PacketLifeTime is 4.096usec * 2^10=4.2msec exactly.
PacketLifeTime	0x0A	
Reserved2	0	

15.2.5.18 MCGROUPRECORD

Table 170 MCGroupRecord

Component	Length(bits)	Offset(bits)	Description
McGroupRID	32	0	RID of this record supplied by Subnet administration in the response to the add (create). add request: value ignored by SA delete request: RID returned on the create
MGID	128	32	Multicast GID for this multicast group add request: if zero, the subnet admin allocates an available MGID, else uses the specified MGID.
Q_Key	16	160	Q_Key supplied in the request add request: non-zero
MLID	16	176	Multicast LID for this multicast group add request: the response provides the MLID

Table 170 MCGroupRecord

Component	Length(bits)	Offset(bits)	Description
MTU	8	192	MTU of this multicast group (For a create must be specified and zero is invalid. For a delete, zero matches all records.) 0:reserved 1: 256 2: 512 3: 1024 4: 2048 5: 4096 6-255: reserved
TClass	8	200	TClass to be used in the GRH if GRH is used; Specified on a create and distributed to the member record on a successful join.
P_Key	16	208	Partition Key for this Multicast group (Must be specified)
RawTraffic	1	224	Traffic will be raw packets (No P_Key) 0-IBA packet traffic (P_Key must be valid) 1-Raw packet traffic
Reserved3	3	225	Reserved (Ignored)
FlowLabel	20	228	Flow label to be used in the GRH if GRH is used; Specified on a create and distributed to the member record on a successful join.
HopLimit	8	248	Hop limit to be used in the GRH if GRH is used; Specified on a create and distributed to the member record on a successful join.

When an entity wishes to create a multicast group, it can be done with either the *SubnAdmConfig()* or the *SubnAdmSet()* methods to create a MCGroupRecord.

o15-3: Using the *SubnAdmSet()* method, the edit modifier (see Table 103 Attributes Common to Multiple Classes on page 618) must be setup accordingly; set to add for creating a multicast group and delete for removing a multicast group. One cannot edit (or modify) a group.

See section 15.4.9 SubnAdmConfig() & SubnAdmConfigResp() - Add, modify or delete RAs on page 745, for specifics on using the *SubnAdmConfig()* method to allocate/delete a MCGroupRecord.

A multicast group can be created by the *SubnAdmSet()* method, specifying Add Record in the edit modifier, the Q_Key, MTU and the P_Key (all other fields are zero). If a particular MGID is required, it can be specified in the *SubnAdmSet()* as well. When a multicast group is deleted, it can be done by the *SubnAdmSet()* method, specifying the Delete Record in the edit modifier and the McGroupRID. Alternatively, it can specify the MGID, Q_Key, MLID, MTU and P_Key. If a field is and can be set to zero for a delete, it will result in a match for that field for all MCGroupRecords by Subnet administration.

o15-4: The addition of a new record or removal of a MCGroupRecord implies that the SM shall program routers and switches with the new multicast information.

15.2.5.19 MCMEMBERRECORD

Table 171 MCMemberRecord

Component	Length(bits)	Offset(bits)	Description
MCMember-RID	32	0	RID of this record Zero specified in the leave request results in a match for all records
MGID	128	32	Multicast GID address for this multicast group Required in the request and returned in the response.
Q_Key	16	160	Q_Key is supplied at Multicast Group Creation time by the creator. Returned in the response.
MLID	16	176	Multicast LID, assigned by the SM at creation time. Zero for a join request. Ignored by Subnet administration. Returned in the response for a join/leave.
LLID	16	192	LID of requester Returned in the response for a join/leave.
TClass	16	208	TClass to be used in the GRH if GRH is used; Specified in the group record and distributed to the member record on a successful join. 0 - unspecified for a query (matches any)
RawTraffic	1	224	Traffic will be raw packets (No P_Key) 0-IBA packet traffic (P_Key must be valid) 1-Raw packet traffic
Reserved3	3	225	Reserved (Ignored)
FlowLabel	20	228	Flow label to be used in the GRH if GRH is used; Specified in the group record and distributed to the member record on a successful join. 0 - unspecified for a query (matches any)

Table 171 MCMemberRecord

Component	Length(bits)	Offset(bits)	Description
HopLimit	8	248	Hop limit to be used in the GRH if GRH is used; Specified in the group record and distributed to the member record on a successful join. 0 - unspecified for a query (matches any)
P_Key	16	256	Partition key is supplied at Multicast Group creation time by the creator. Non-zero in the request. Checked by Subnet administration. (Note: Multicast groups can't span partitions with a single MLID)
Reserved	4	272	Reserved -- shall be set to 0 (zero)
MCSL	4	276	SL to be used for this MC group

When an entity wishes to join a multicast group, it can be done with either the *SubnAdmConfig()* or the *SubnAdmSet()* methods to create a MCMemberRecord.

When using the *SubnAdmSet()* method, the edit modifier (see Table 103 Attributes Common to Multiple Classes on page 618) must be setup accordingly; add for joining a multicast group and delete for leaving a multicast group.

o15-5: SA shall respond to a *SubnAdmSet()* method for a MCMemberRecord that has the edit modifier set to edit with a *SubnAdmGetResp()* with the status set to invalid attribute.

A multicast group can be joined using the *SubnAdmSet()* method by specifying Add Record in the edit modifier and the MGID (all other fields are zero). See section 15.4.9 SubnAdmConfig() & SubnAdmConfigResp() - Add, modify or delete RAs on page 745, for details using the *SubnAdmConfig()* method for joining a multicast group.

When leaving a multicast group, the *SubnAdmSet()* method can also be used by specifying the Delete Record in the edit modifier and the MCMemberRID. Alternatively, one can specify the LLID, MGID, Q_Key, MLID, MTU and P_Key. If a field is set to zero for a delete request, it will result in a match for that field in all the MCMemberRecords by Subnet administration. (Note: One could leave all the Multicast groups with one *SubnAdmSet()* request by specifying just the LLID.)

o15-6: An addition of a new MCMemberRecord or removal of a MCMemberRecord implies that the SM shall program routers and switches with the new multicast information.

15.2.5.20 GUIDINFORECORD

Table 172 GuidInfoRecord

Component	Length(bits)	Offset(bits)	Description
GuidInfoRID	32	0	RID of this record
NodeRID	32	32	RID of related NodeInfo record
GUIDInfo	512	64	Content of GUIDInfo attribute record

15.2.5.21 SARESPONSE

Table 173 SAResponse

Component	Length(bits)	Offset(bits)	Description
EndOfTableRIDs	32	0	Byte Count to the end of table RIDs, i.e., beginning of bulk data.
Current Key	32	32	Current SA key
TableRID	variable, each entry is 48 bits	64	See Table 174 TableRID Component on page 727 below for format description.
BulkData	0 to limit	varies	Bulk data

Table 174 TableRID Component

SubComponent	Length(bits)	Offset(bits)	Description
AttributeID	16	0	Attribute Identifier of attribute record type
EndOfTable	32	16	Offset to the end of records with this AttributeID

The SAResponse record is utilized by SubnAdmGetBulkResp() exclusively in the first packet response to indicate contents of the following datastream from the SA.

15.3 RELIABLE MULTI-PACKET TRANSACTION PROTOCOL

C15-10: The Multiple-packet transaction protocol specified in this section, 15.3 Reliable Multi-Packet Transaction Protocol on page 727, shall be used for *SubnAdmGetBulk(), SubnAdmGetTable(), and SubnAdmConfig()* transactions, whenever their request or response spans multiple packets

15.3.1 SUBNET ADMINISTRATION MAD DATA FIELD USAGE

The following fields of Subnet Administration MAD data field are used to support the multi-packet protocol.

15.3.1.1 Segment Number Field

The Segment Number field identifies the relative position of each packet within a multipacket request or response. Segment Numbers for multi-packet requests and responses begin at segment number 1; the segment number of a single packet request or single-packet response is set to 0.

For a description of the usage of the segment number field in acknowledgment packets, resend request packets, and KeepAlive packets, see 15.3.1.3 Fragment Flag on page 728.

15.3.1.2 Payload Length

The payload length field is valid only for the first packet of a multipacket SubnAdmConfig() request, and the first and last packet of multi-packet SubnAdmGetBulk(), SubnAdmGetTable() responses. In all other packets of a multipacket request or response, the payload length field is reserved.

In the first packet of a SubnAdmConfig() request, the value in the payload length field indicates the sum of the lengths in bytes of the Admin Data fields in all packets which the requester is sending for the transaction.

In the first packet of a multi-packet SubnAdmGetBulk(), SubnAdmGetTable() or SubnAdmConfig() response, the payload length field indicates the expected sum of the lengths of the Admin Data fields in all packets of the entire multipacket response.

In the last packet of a multipacket SubnAdmGetBulk(), SubnAdmGetTable() or SubnAdmConfig() response, the payload length field indicates the number of valid bytes in the Admin Data field.

After the first response to a SubnAdmGetBulk(), SubnAdmGetTable() request is sent, the actual payload length may change. The payload length field in the last response packet indicates the number of valid bytes in the Admin Data field, and the Last Packet bit (bit 2) of the fragment flag is set to one.

15.3.1.3 Fragment Flag

The fragment flag identifies the packet as either the first packet of a request or response, a midstream packet, or a the last packet. The fragment flag is also used to specify other characteristics of the packet as indicated in Table 175 on page 729. For single-packet request/response transactions, the fragment flag is set to zero.

Table 175 Fragment Flag Description

Bit	Name	Description
0	First Packet:	Set to one in the first packet of a multi-packet request and the first packet of a multi-packet response; set to zero otherwise.
1	Reserved	
2	Last Packet:	Set to one in the last packet of a multi-packet request and the last packet of a multi-packet response; set to zero otherwise.
3	Resend Request:	For a requester, this bit is set to one to request the responder to restart sending packets for a multi-packet response beginning with the Segment Number indicated in the Segment Number field. For a responder, this bit is set to one to restart sending packets of a multi-packet request beginning with the Segment Number indicated in the Segment Number field. In all other cases, bit 3 is set to zero. For resend request packets, all fields in the MAD data field other than the fragment flag and segment number are ignored by the recipient.
4	Reserved.	
5	Acknowledgment Packet:	For a requester, this bit is set to one to acknowledge the receipt of all response packets up to and including the packet with the segment number indicated in the Segment Number field. For a responder, this bit is set to one to acknowledge the receipt of all request packets up to and including the packet with the segment number indicated in the Segment Number field. After receipt of a packet is acknowledged, resources associated with the packet are released and a request to retransmit the packet is not allowed. (See bit 3 above.) The segment number intervals at which acknowledgment packets are sent is not specified. For acknowledgment packets, all fields in the MAD data field other than the segment number, fragment flag, and window are ignored by the recipient.
6	Reserved	
7	KeepAlive Packet:	Set to one by the sender of a multipacket request or response to request the recipient to re-initialize the timer for the transaction to the segment timeout period. The sender may be either the requester, as in the case of a multipacket request, or the responder, as in the case of a multipacket response. For the keepAlive packet, all fields in the MAD data field other than the fragment field are ignored by the recipient.

Table Table 176 on page 730 gives examples of the use of the fragment flag.

Table 176 Fragment Flag Usage Examples

Type of Transaction	Packet	bits 7:0
Single Packet Request	only packet	00000000
Multi-packet Request	first packet	00000001
	nth (not last) packet	00000000
	last packet	00000100
	Acknowledgment packet sent by responder	00100000
	Resend Request sent by responder	00001000
Single Packet Response	only packet	00000000
Multi-packet Response	first packet	00000001
	nth (not last) packet	00000000
	last packet	00000100
	Acknowledgment packet sent by requester	00100000
	Resend Request sent by requester	00001000

15.3.1.4 WINDOW

The window field is valid only in a multipacket SubnAdmGetBulk() or SubnAdmGetTable() request, and in an acknowledgment packet of a SubnAdmConfig() transaction.

The window value in a multipacket SubnAdmGetBulk() or SubnAdmGetTable() request specifies the number of packets the responder may send before it receives an acknowledgment packet. When an acknowledgment packet is received by the responder, a number of additional packets (subsequent to the packet being acknowledged) equal to the window value may be sent.

For a SubnAdmConfig() transaction, the window value in acknowledgment packets indicates the number of packets which the requester may send (subsequent to the packet being acknowledged) before it receives another acknowledgment packet. For the first burst of request packets of a SubnAdmConfig() transaction, the default window value of 64 is used by the requester. If the window value field in any acknowledgment packet is less than the default window value of 64, the default window value of 64 is used by the recipient.

15.3.1.5 ADMIN DATA

When sending a packet during a multipacket request or response, the admin data fields of all the packets contain the full 192 bytes of data except for the last packet. The last packet of the multipacket request or response may contain fewer bytes as indicated by the payload length field. (See 15.3.1.2 Payload Length on page 728.)

15.3.2 TIMEOUTS

The multi-packet protocol uses two timeout values which are specified in port attributes:

- PortInfo:SubnetTimeout: This the maximum delay time from a port to any other port in the subnet.
- SA ClassPortInfo:ResponseTimeValue -
  - The maximum expected time within which the SA agent of a given port will respond to an SA request.
  - The maximum expected time within which an SA agent of a given port will acknowledge the receipt of a packet after it receives the last packet allowed by the window parameter during a multi-packet transaction.
  - The maximum expected time interval between the sending of subsequent packets of a multipacket SubnAdmConfig() request or between the sending of subsequent packets of a multipacket SubnAdmGetTable(), SubnAdmGetBulk(), or SubnAdmConfig() response.

The above timeouts are used to calculate the following timeout periods:

15.3.2.1 RESPONSE TIMEOUT PERIOD

When a packet is sent which requires a response, the response timeout period is the maximum expected time within which the sender expects to receive the response. The response timeout period is used for the following responses:

- For the sender of a SubnAdmGetTable() or SubnAdmGetBulk request, the response timeout period is the maximum time within which the requester expects to receive the first response to the request.
- For the sender of a multi-packet request or multipacket response which contains more packets than allowed by the Window field, the response timeout period is the maximum time within which the sender expects to receive an acknowledgment packet after it sends the

last packet allowed by the window parameter. When an acknowledgment packet is expected but not received, all resources associated with the transaction may be released.

The value of the response timeout is equal to 2*(PortInfo:SubnetTimeout of the local port) + ClassPortInfo:ResponseTimeValue of the remote port.

15.3.2.2 Segment Timeout Period

The segment timeout period is the maximum expected time between the receipt of subsequent packets of a multipacket SubnAdmConfig() request, or a multipacket SubnAdmGetTable() or SubnAdmGetBulk() response.

The recipient of the multipacket request or response starts the segment timeout period for the transaction whenever it receives a packet for the transaction. The segment timeout period is reinitialized and restarted whenever a packet is received as long as there are outstanding packets expected. The KeepAlive packet will also reset the segment timer as any other response packet.

The value of the segment timer is equal to PortInfo:SubnetTimeout of the multipacket recipient + ClassPortInfo:ResponseTimeValue of the multipacket sender.

15.3.3 Reliable Multi-packet Protocol Description

The following two sections describe the reliable multi-packet protocol.

15.3.3.1 Multi-Packet Protocol: Multi-Packet Response

Figure 168 on page 733 shows a request/response transaction in which the response contains multiple packets. SubnAdmGetTable() and SubnAdmGetBulk() transactions use this protocol.

In Figure 168, the requester initiates the multi-packet protocol by sending a request with the fragment flag set to b'10000000' and the window field set to n. When the request packet is sent, the requester initializes the response timer for the transaction. (See 15.3.2.1 Response Timeout Period on page 731.) If the responder cannot return a response within a time period equal to the ClassPortInfo:ResponseTimeValue of the port on which the request was received, it responds with a KeepAlive packet. Additional KeepAlive packets may be sent if additional time is required to send the first response packet.

If the requester does not receive a response packet after the response timer expires, a response timeout error is recognized. See 15.3.4 Error Handling on page 736.

When the responder has the response data, it sends the first n response packets. Each packet is sent within a time period of the ClassPortInfo:Re-

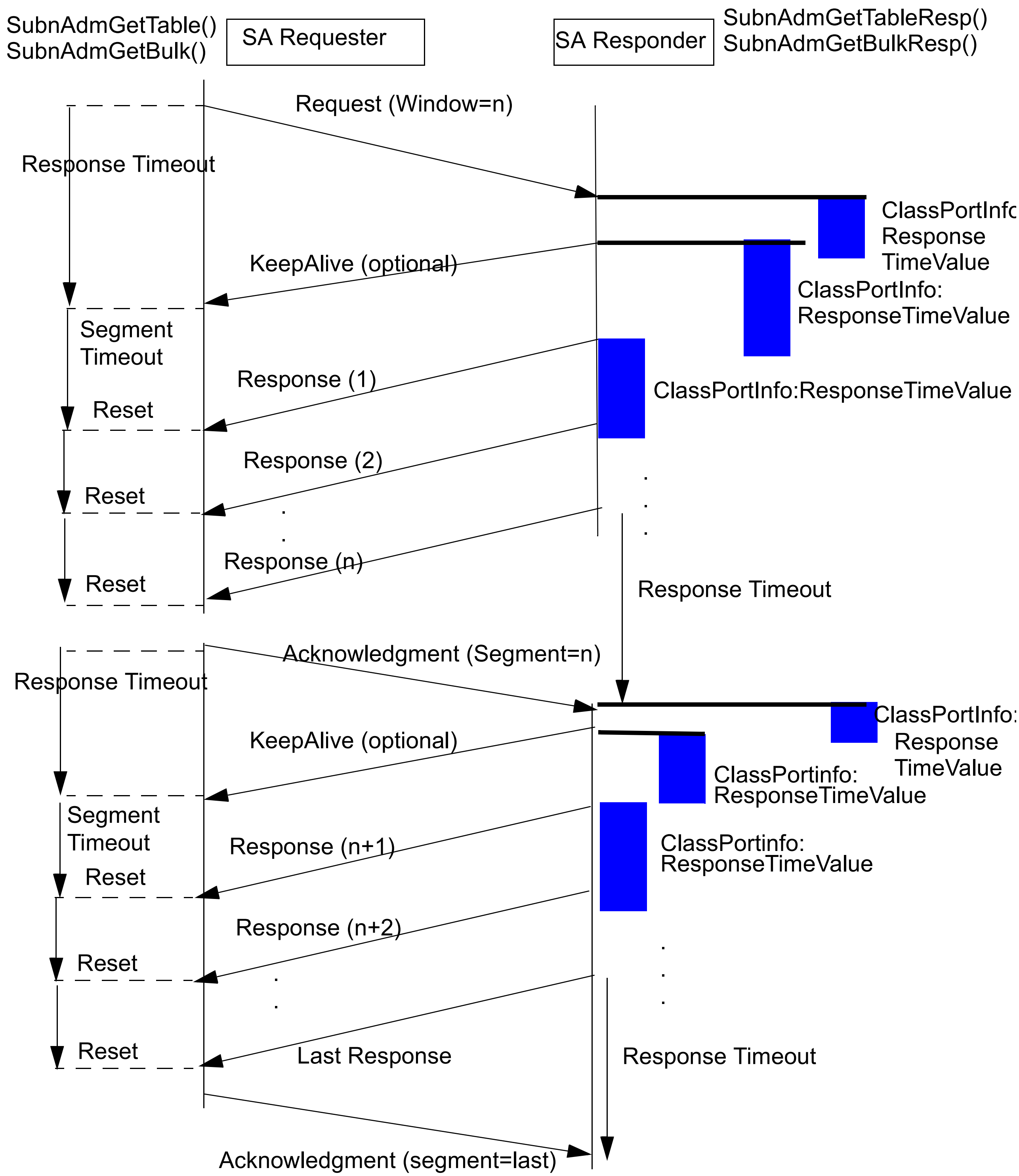


Figure 168 Multi-Packet Protocol: Multi-packet response

sponseTimeValue of the time when the previous packet was sent. Except for the optional KeepAlive packet, the segment number of each response packet increments starting from one.

Whenever the requester receives a response, it initializes the segment timer for the transaction. (See 15.3.2.2 Segment Timeout Period on page 732.) The segment timer is reinitialized and restarted whenever a response packet or KeepAlive packet is received as long as there are outstanding responses expected. The segment timer is stopped when there are no outstanding responses expected. If the segment timer expires, a segment error is recognized. See 15.3.4 Error Handling on page 736.

The responder continues sending packets until n packets have been sent. Since there are additional packets to be sent for the response, the fragment flag of the nth packet is set to b'00000000', to indicate that the packet is not the last packet. After sending the nth packet, the responder initializes the response timer waits for an acknowledgment packet.

After receiving the nth packet, the requester sends an acknowledgment packet with a fragment flag equal to b'00100000' and a segment number equal to n. This packet acknowledges the receipt of the first n packets and allows the responder to release resources associated with them. Upon receipt of this acknowledgment packet, the responder continues sending response packets for the request beginning with segment number n+1. If no acknowledgment packet is received when the response timer expires, the responder may release resources associated with the transaction.

If the responder sends the last response packet before sending the next n packets, it sets the fragment flag to b'00000100' to indicate that the last packet has been sent. The Payload length field of the last packet is set to the number of valid data bytes present in the Admin Data field of the last packet.

When sending the last packet, the responder initializes the response timer. The responder retains resources associated with unacknowledged packets until the last packet is acknowledged or the response timer expires because the requester may send a resend-request packet to recover lost packets. If the responder does not receive an acknowledgment before the response timer expires, the responder releases all resources for the request.

15.3.3.2 MULTI-PACKET PROTOCOL: MULTI-PACKET REQUEST

Figure 169 on page 735 shows a request/response transaction in which the request contains multiple packets. SubnAdmConfig() transactions use this protocol.

In the figure, the requester initiates the transaction by sending the initial

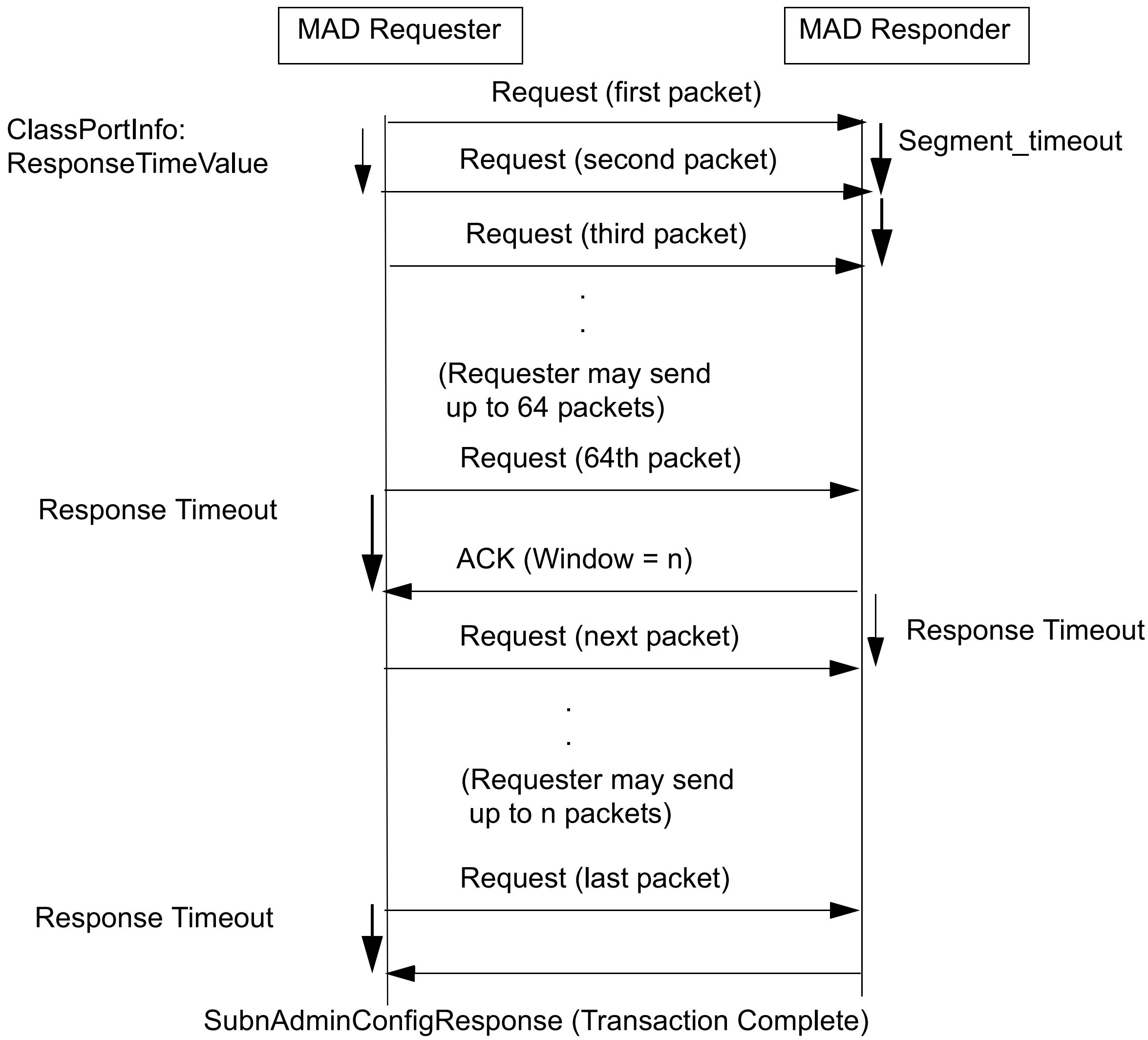


Figure 169 Multipacket Protocol: Multipacket Request

request with a segment number of 1. The payload length field is set to sum of the lengths in bytes of the Admin Data fields in all packets to be sent for the request. The requester then sends subsequent packets within a time period of ClassPortInfo:ResponseTimeValue of the previous packet. The segment number of each packet is incremented by one. The requester may send 64 packets before receiving the first acknowledgment packet.

As the multipacket request is received, the recipient checks to ensure that the segment number of each packet received is one higher than the previous packet received. If a packet is received with a segment number which is not one higher than the previous packet, a segment error is recognized. See 15.3.4 Error Handling on page 736. The recipient also initializes the segment timer whenever it receives a packet. (See 15.3.2.2 Segment Timeout Period on page 732.) If the segment timer expires, a segment error is recognized. See 15.3.4 Error Handling on page 736.

In the example shown, the requester sends 64 packets, which is the default number of packets, and does not receive an acknowledgment packet. When sending the 64th packet, the requester initializes the response timeout period and waits for an acknowledgment packet.

When the responder has sufficient buffer space available, it responds with an acknowledgment packet with the window value of n. This indicates to the requester that it may send n additional packets (subsequent to the packet being acknowledged). Upon receipt of this acknowledgment packet, the requester may send up to n additional packets.

In the example shown, the requester finishes sending all of the packets of the multi-packet request before sending n additional packets. The last packet sent has a fragment flag of b'00000100', indicating that the packet is the last packet of the request. When sending the last packet, the requester initializes the response timer. The requester retains resources associated with unacknowledged packets until the SubnAdmConfigResp() is received or the response timer expires; this is necessary because the responder may send a resend-request packet to recover lost packets. If the requester does not receive a resend-request packet or the SubnAdmConfigResp() before the response timer expires, the requester recognizes a response timeout error.

When the responder has received all of the packets, it sends a SubnAdminConfigResp() packet to indicate that the transaction is complete. No other acknowledgment packet is sent.

15.3.4 Error Handling

The following errors and the associated recovery are given:

15.3.4.1 Response Timeout Error

A response timeout error is recognized when the response timer expires. See 15.3.2.1 Response Timeout Period on page 731 for a definition of the response timeout period.

The recovery for a response timeout error is to resend the packet for which the response was expected, provided a vendor-specific number of retries have not been performed. When the maximum number of retries has been performed, resources associated with the transaction may be released.

The maximum number of times a packet may be resent is 7.

15.3.4.2 SEGMENT ERROR

A segment error is recognized when the segment timer expires or when midstream packet is received during a multipacket transaction whose segment number is not one greater than the previous packet received.

The error recovery for a segment error is to 1) discard packets received with segment numbers greater than the segment number of the missing packet, 2) send a resend-request packet with the fragment flag set to b'00010000'. The segment number field of the resend request packet is set to the segment number of the packet which was expected but not received. This causes packet transmission to restart beginning with the packet whose segment number is equal to the segment number of the resend-request packet. When the resent packet with the requested segment number is received, the recipient stops discarding packets.

If a resend-request packet is received which contains a segment number of a packet for which resources have been released, status indicating an invalid attribute field is returned.

The number of resend requests is vendor-specific with the limitation that the maximum number of resend requests which may be sent for a specific segment error is 7. When the vendor-specific number of resend requests have been sent, resources associated with the transaction may be released.

15.4 OPERATIONS

This section describes the operational aspects of SA.

15.4.1 RESTRICTIONS ON ACCESS

There are two types of access restrictions involved in SA: Authenticating the requestor of information, and restricting the data that the requestor is allowed to receive. These are discussed below.

15.4.1.1 AUTHENTICATING THE REQUESTOR

The P_Key index in a request MAD is used by SA to authenticate the sender of a request, along with the GID and LID of the request MAD. This information can then be used to determine if, for example, the sender is or is not a valid (possibly standby) SM.

15.4.1.2 ACCESS RESTRICTIONS FOR PATHRECORDS

C15-11: Subnet Administration shall return to a requester only path records for which the source port, destination port, and requester all share a P_Key pairwise. See the remainder of this section (15.4.1.2) for a detailed explanation.

- Two ports share a P_Key when there is at least one valid P_Key in one port's P_Key Table that matches a P_Key in the other port's P_Key Table. (See 10.9.3 Partition Key Matching on page 457 for the definition of P_Key matching, and 10.9.1.2 Special P_Keys on page 455 for the definition of a valid P_Key.)
- "Pairwise" means that P_Key sharing must be present between three pairs of ports: path source port and path destination port; path source port and a port of the requester; path destination port and a port of the requester. Each of the three matches may be based on the matching of different P_Keys.
- The path source and destination ports used to determine sharing are the ones that are implicit in the SGID (or SLID) and DGID (or DLID) of the path.
- The port of the requestor that is used to determine pairwise sharing may be any port of the node from which the request came. The path record should be returned if any port(s) of the requesting node, not just the port from which the request MAD came, provides the pairwise sharing described above. The requestor port sharing a P_Key with the source port need not be the same port as the requestor port sharing a P_Key with the destination port.
- All ports involved in this determination must be on the subnet administered by the Subnet Administrator to which the request is directed.

15.4.1.3 ACCESS RESTRICTIONS FOR OTHER ATTRIBUTES

C15-12: When a requester node requests information from the Subnet Administrator about a subject node, the Subnet Administrator shall return only information about subject nodes for which the requester shares a P_Key, with exceptions noted below at **Exceptions.**

Sharing is defined as follows:

- Two ports share a P_Key when there is at least one valid P_Key in one port's P_Key Table that matches a P_Key in the other port's P_Key Table. (See 10.9.3 Partition Key Matching on page 457 for the definition of P_Key matching, and 10.9.1.2 Special P_Keys on page 455 for the definition of a valid P_Key.)
- The port of the requestor or the subject node that is used to determine sharing may be any port of either the requester node or the subject node. The information should be returned if any port(s) of the requesting node, not just the port from which the request MAD came, provides the sharing described above.
- All ports involved in this determination must be on the subnet administered by the Subnet Administrator to which the request is directed.

Exceptions: PortInfoRecords are always provided with the M_KEY component set to 0, except in the case of a trusted subnet manager; in that case the actual M_KEY component contents shall be provided. Trust of other subnet managers is implied by earlier provision of a valid SM_KEY previously by the requester during the operations leading to the establishment of the SM master on the subnet. PortInfoRecords with complete M_KEY information shall be openly shared between trusted SMs. PartitionRecords shall be openly shared between trusted SMs.

15.4.2 Locating Subnet Administration

C15-13: It shall be possible to determine the location of SA from any endport by sending a GMP to QP1 (the GSI) of the node identified by the endport's PortInfo:MasterSmLID, using in the GMP the base LID of the endport as the SLID, the endport's PortInfo:MasterSMSL as the SL, the well-known Q_Key (0x8001_0000), and whichever of the default P_Keys(0xFFFF or 0x7FFF) was placed in the endport's P_Key Table by the SM (Table 143 Initialization on page 695).

C15-14: A SubnAdmGet(ClassPortInfo) sent according to C15-13 shall return all information needed to communicate to Subnet Administration. Alternatively, valid GMPs for the SA sent according to C15-13 shall either return redirection responses providing all such information, or shall be normally processed by the SA.

15.4.3 Versioning

C15-15: SA_KEY shall be provided by the subnet administration methods to serve as a *versioning* key, in which the SA_KEY value provided is compared with a previous SA_KEY, and the SA can optionally provide only records added since the provided key was issued.

Simple SAs will provide all records regardless of key value. This works for *SubnAdmGetBulk()* and *SubnAdmGetTable()* methods.

15.4.4 Event Forwarding Subsystem

15.4.4.1 SubnAdmInform(), SubnAdmInformResp(), SubnAdmReport(), & SubnReportResp()

The event forwarding subsystem exists for the purpose of subscribing for the subnet management class of traps from the SA.

C15-16: Event forwarding operations directed at SA shall conform to the common methods as described in 13.4.5 Management Class Methods on page 607.

15.4.5 Administration Query Subsystem

15.4.5.1 Component Mask

In the administration query subsystem the 64 bit component mask in the SA MAD is used in query operations to specify particular attribute components to query on. The component mask can refer to only an entire component, not elements or parts of a component.

C15-17: In the component mask, for query operations the 0 bit must refer to the first component, the 1 bit must refer to the second element, and so forth.

C15-18: When a component mask bit is set to 1 in a query, the component must be matched in all responses to a query operation. When a component mask bit is set to 0 in a query, all records otherwise matching must be returned regardless of the setting of that component. (There is an exception for PathRecords; see 15.2.5.17 PathRecord on page 717.)

C15-19: For edit operations the component mask shall be used to refer to a specific component to be edited. Such editing is only valid using the *SubnAdmConfig()* method. As with query operations, the bits all map to specific components to be edited, the 0 bit refers to the first component, the 1 bit refers to the second component, and so on. If the component mask bit for a component is set to 0, that component must be ignored in the edit operation: no change is made to that particular attribute component.

A component mask of all ones means that all components are to be used for a query. For an edit, a component mask of all ones means that all components are to be edited. The result of using a component mask of 0 for a query or an edit operation is undefined.

For the event forwarding subsystem the component mask is unused.

15.4.5.2 ATTRIBUTE AND ATTRIBUTE MODIFIER USE

Query and editing of tables (groups of RAs) is done with the use the Attribute ID and Attribute Modifier fields.

The Attribute ID is used to reference the table to query or edit. In bulk operations the Attribute ID is unused, and set to 0.

The Attribute Modifier and End RID are used to indicate ranges of RAs based on a RA RID (not the related RID).

See 15.2.5 Attributes on page 709 for more information on RAs.

15.4.6 SUBNADMGETTABLE() & SUBNADMGETTABLERESP()

SubnAdmGetTable() is used to request an RA table. Operations are allowed on a specific table only, specified by attribute identifier.

15.4.6.1 QUERY BY TEMPLATE

C15-20: The Attribute modifier of 0xFFFFFFFF shall indicate a query by template rather than by RID.

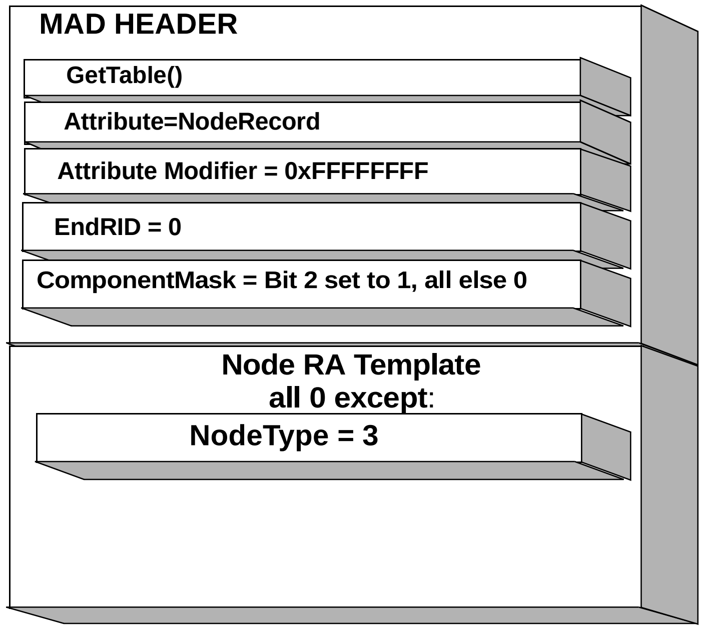


Figure 170 Search by NodeRecord Template to search for all routers

Consequently, an attribute modifier of 0xFFFFFFFF cannot be specified for query of RID values.

A query by template uses the RA specified by the Attribute identifier (also called the Attribute ID) to determine the query format contained in the SA MAD data area of the request. Such requests for an exact match of a single component, such as a ServiceRecord:ServiceName is indicated with that value being set to the value to be searched for, the component mask being set to indicate the ServiceName component as the attribute component to be searched for, and all other values in the template are ignored.

- Attribute Modifier set to 0xFFFFFFFF - If the attribute modifier is set to 0xFFFFFFFF, this is interpreted as a query indicated by the RA(s) following in the data area.
- Component set to exact value or values to match
- All other component values are ignored, and may be set to 0.
- Deleted RAs are indicated with the major RID intact, but all values set to 0 in that RA

A a query by template for NodeRecords with a NodeType of 3 would supply the data shown in Figure 170 on page 741 and Table 177 on page 742 in the request:

Table 177 SubnAdmGetTable query for all NodeRecords with a specific NodeType

Component	Value	Interpretation
Header:AttributeID	0x0011	Specify NodeRecords.
Header:Attribute Modifier	0xFFFFFF FF	Query specified in attribute header.
Header:EndRID	0x0000	Value is ignored by SA.
Header:ComponentMask	0x2	Set bit 2 to one, all else to 0
NodeRecord:NodeType	3	Obtain all NodeRecords with NodeType value of 3.
All other NodeRecord components	0	Ignore this field.

15.4.6.2 QUERY BY RID RANGE

C15-21: A SubnAdmGetTable() with an attribute modifier not equal to 0xFFFFFFFF shall indicate a query by RID range.

C15-22: A query by RID range shall use the attribute modifier as the start of the RID range, and the EndRID component as the end of the range returned, inclusive.

The attribute value indicates the type of record returned in a query by RID Range. (For example, for the NodeRecord RA, the Attribute ID value is 0x0011).

The two following tables specify an example of a query, and a range request.:

Table 178 SubnAdmGetTable query for all NodeRecords within a given range

Component	Value	Interpretation
Header:AttributeID	0x0001	Specify NodeRecords.
Header:Attribute Modifier	0x003000	Get records starting at NodeRID 0x30 or greater.
Header:EndRID	0x004000	Get records ending at NodeRID 0x40 or below.
Header:ComponentMask	0x0	Data area is set to 0, ignored by SA
AllNodeInfo components	0	Data area is all set to 0, ignored by SA

15.4.6.3 Requesting all records of a table

To request all records of a table, the Attribute ID is set for the table desired, the SA_KEY is set to 0, and the Attribute modifier is set to 0 and EndRID value is set to 0xFFFFFFFF.

15.4.6.4 Requesting all new table records since last request

C15-23: A *SubnAdmGetTable()* with an SA_KEY of 0 shall return the entire table specified by the request, with the current value of the SA_Key.

C15-24: A *SubnAdmGetTable()* with a non-zero SA_Key that was current at a prior time shall return either the changes made to the table specified by the request since the provided SA_Key was current; or the entire table specified by the request; with the current value of the SA_Key.

In this way the SA_KEY can be used to limit the records returned to only the records added since last query.

C15-25: A *SubnAdmGetTable()* with a non-zero SA_Key that has never been current will return an empty response (no records) with a status field indicating invalid attribute.

15.4.7 SubnAdmGetTableResp()

Subnet Administration uses the *SubnAdmGetTableResp()* to respond to all *SubnAdmGetTable()* queries.

o15-7: SA may indicate a refused request by returning a *SubnAdmGetTableResp()* with the status field providing the reason for refusal.

A *SubnAdmGetTable()* and the corresponding *SubnAdmGetTableResp()* is illustrated in Figure 171 on page 744. Subsequent *SubnAdmGetTableResp()* MADs of this transaction will have the *FragmentFlag and Segment Number* values set to indicate place in the data stream. The continuation of the data to is reassembled by the requester in its own data area.

The EndRID value for *SubnAdmGetTableResp()* is undefined.

The RAs for a *SubnAdmGetTableResp()* for the example of Figure 171 on page 744 are contained within a SA MAD as shown. Note that records may be broken across successive response MADs.

15.4.8 *SubnAdmGetBulk() & SubnAdmGetBulkResp()*- Bulk Table Retrieval

o15-8: If implemented, *SubnAdmGetBulk()* shall return all records currently held by SA.

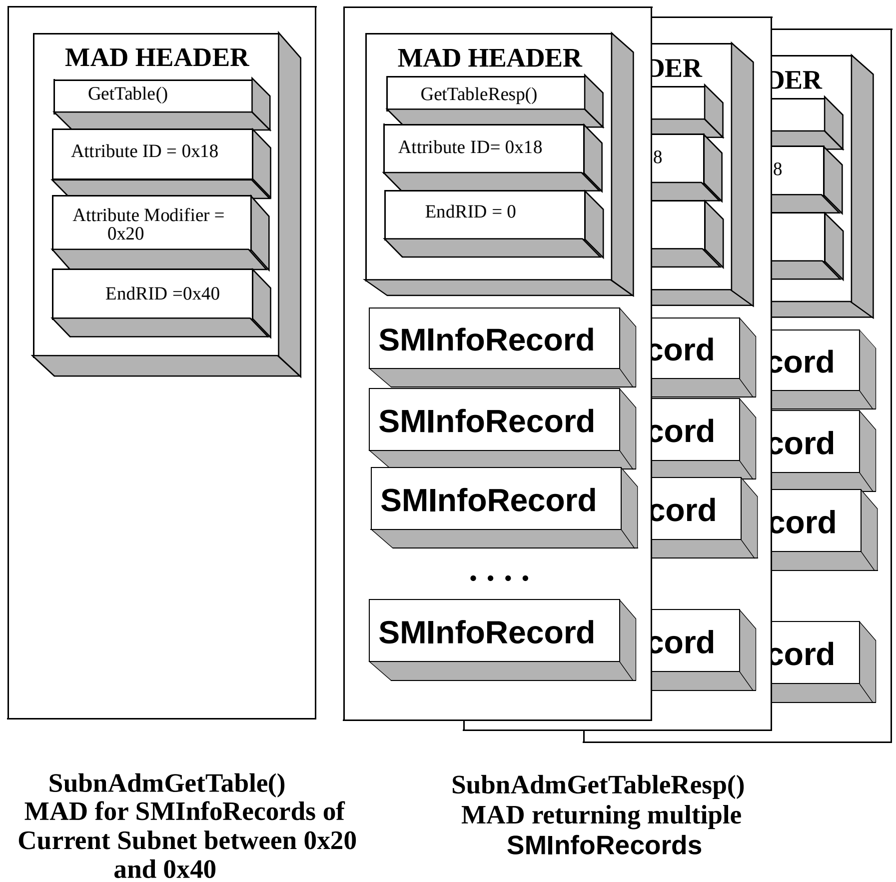


Figure 171 Example GetTable(), GetTableResponse() Layout

SubnAdmGetBulk() has no implied attribute; the data payload is all set to zeroes, and is ignored on receive. The Attribute Modifier is ignored by the receiver.

o15-9: *SubnAdmGetBulkResp()* MADs shall have a data area consisting of the SAResponse Attribute, followed by the actual record tables.

o15-10: If SA has no data, *SubnAdmGetBulkResp()* shall return with the Attribute Modifier field is set to all zeros, and the data area is set to all zeros.

o15-11: *SubnAdmGetBulk()* shall use the SA_Key with the same semantics as *SubnAdmGetTable()*.

An example of a *SubnAdmGetBulkResp()* is illustrated in Figure 172 on page 745.

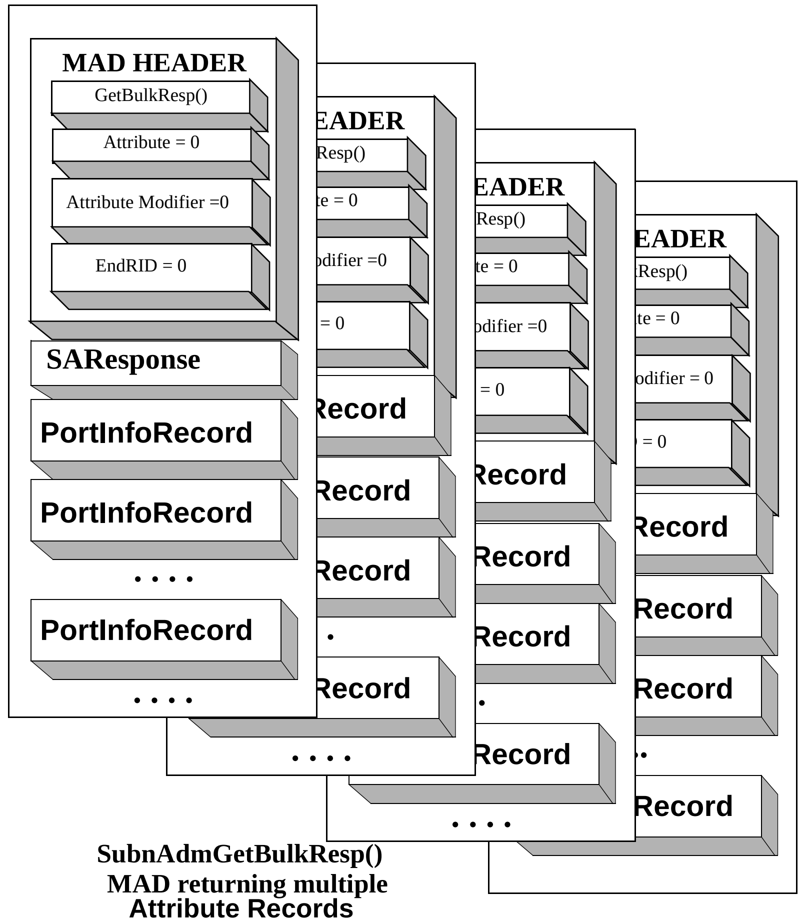


Figure 172 Example GetBulkResp() Layout

15.4.9 *SubnAdmConfig() & SubnAdmConfigResp()* - Add, modify or delete RAs

SubnAdmConfig() provides the ability to add, delete, or modify entire SA tables, in contrast to *SubnAdmSet()*, which only operates on single records. The RAs supplied with the *SubnAdmConfig()* are the RAs to be operated on, subject to component mask settings. The RIDs in those RAs indicate the RAs to be operated on.

o15-12: State records (see Table 148 Subnet Administration Attributes (Summary) on page 710) shall only be modified by *SubnAdmConfig()s* sent from the master subnet manager.

Hence the target of a *SubnAdmConfig()* can only be a non-master subnet manager. A use for this is for a master SA to “push” data to a standby.

o15-13: If the number of records supplied in a SubnAdmConfig() is fewer than the number to fill the RID range specified by the attribute modifier and the EndRID, the final record sent is repeated until the end of the range.

Note that the previous paragraph allows editing the attributes of a range of records by sending a single record, since the component mask can be set to modify only the specific desired components.

15.4.9.1 SUBNADMCONFIGRESP()

o15-14: *SubnAdmConfigResp()* shall contain an admin data field set to all 0s, and a status field indicating the success or failure of the corresponding *SubnAdmConfig()*.

15.4.10 SUBNADMGET() & SUBNADMGETRESP(): GET AN RA

C15-26: In response to a *SubnAdmGet()* with a non-zero RID contained in the attribute modifier, *SubAdmGetResp()* shall return the RA with that RID, subject to the access rules specified in 15.4.1 Restrictions on Access on page 737.

C15-27: In response to a *SubnAdmGet()* with 0xFFFFFFFF in the attribute modifier, *SubAdmGetResp()* shall return the RA matching the components supplied in the component mask, subject to the access rules specified in 15.4.1 Restrictions on Access on page 737.

C15-28: The SA_Key value provided in a *SubnAdmGet()* must be ignored, and the SA_Key returned in a *SubAdmGetResp()* shall be zero.

o15-15: If more than one RA would be returned as a result of matching, *SubAdmGetResp()* shall return a status of ERR_REQ_INVALID.

Table 179 SubnAdmGet query for a NodeRecord on page 746 shows an example *SubnAdmGet()* query for a NodeRecord.

Table 179 SubnAdmGet query for a NodeRecord

Component	Value	Interpretation
Header:AttributeID	0x0011	Specify NodeInfo records.
Header:Attribute Modifier	0xFFFFFFFF	Signifies query using data matching in supplied attribute.
Header:EndRID	0x0000	Set to 0, ignored by SA.
NodeInfo: PortGUID	0x0004	Obtain NodeRecord with PortInfoGID value of 4.
All other NodeInfo components	0x0000	All set to 0, SA will Ignore these fields.

15.4.11 SUBNADMSET(): SET AN RA

C15-29: In response to a *SubnAdmSet()* with an edit modifier of add and a zero RID in the attribute modifier, the RA contained in that MAD will be added to the SA, and a *SubAdmGetResp()* shall be returned containing the RA provided along with the RID of that RA, subject to the access rules specified in 15.4.1 Restrictions on Access on page 737.

C15-30: A *SubnAdmSet()* with an edit modifier of edit shall not be performed.

C15-31: If a *SubnAdmSet()* with an edit modifier of delete is received by the SA containing the entire attribute record to delete, with all components matching, and the attribute modifier is set to the value of the major RID, then that RA shall be deleted and *SubAdmGetResp()* is returned with a zero status value, provided that the requestor is allowed access to that record according to the access rules specified in 15.4.1 Restrictions on Access on page 737.

o15-16: If *SubnAdmSet()* with an edit modifier of delete is in any way ambiguous to the SA, or access to it would violate access rules, a null response must be returned to the requester with a status of ERR_REQ_INVALID.

CHAPTER 16: GENERAL SERVICES

This chapter describes the range of management services that the IBA provides under general services, except for the Subnet Administration which is described in the previous chapter. General management services provide the following management classes:

- Performance Management - provides methods that enable a manager to retrieve performance statistics and error information from IBA components.
- Baseboard Management - provides a means to transport messages to components beyond the subnet, to "out of band" components. An example might be to chassis temperature monitoring and control hardware on an IBA channel adapter.
- Device Management - provides the means to perform I/O controller / I/O unit management. This class defines the mechanisms to send and receive device management packets between two subnet-attached points, typically between an HCA and a TCA. The TCA provides an interface to the I/O controller and I/O device.
- SNMP Tunneling - provides a set of methods, data formats and attributes to support SNMP tunneling. The SNMP packet is embedded in the IBA-compliant management datagram.
- Vendor Specific - provides a set of general purpose methods. Vendors are free to define new methods and attributes, however they conform to management datagram formats and restrictions described herein.
- Application Specific - provides a set of general purpose methods. Applications are free to define new methods and attributes, however they conform to management datagram formats and restrictions described herein.
- Communication Management - provides the mechanisms to establish, terminate, and migrate connections between nodes, and provides basic service ID resolution.

16.1 PERFORMANCE MANAGEMENT

C16-1: The Performance Management Agent is mandatory on all nodes.

The Performance Management class provides mechanisms to enable a performance management entity to retrieve performance and error statistics from InfiniBand components. Performance quantities are divided into two classes:

- Mandatory for all ports of all nodes (TCAs, HCAs, Switches, and Routers). These quantities are deemed necessary to support fundamental instrumentation and performance analysis of a multi-vendor InfiniBand fabric
- Optional. These quantities may be implemented at the vendor's discretion, and are described here as an aid to standardization.

16.1.1 MAD FORMAT

C16-2: The datagrams in the Performance class shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 173 Performance Management MAD Format on page 749 and Table 180 Performance Management MAD Fields on page 749 below.

Figure 173 Performance Management MAD Format

bytes	
0	Common MAD Header
...	
20	
24	Reserved
...	
60	
64	Data
...	
252	

Table 180 Performance Management MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 604
Reserved	40 bytes	This field is reserved and shall be set to zeroes.
Data	192 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.1.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. No class-specific bits are defined.

Table 181 Performance Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617
8-15	-	Class-specific bits are reserved

16.1.2 METHODS

The Performance Management class uses a subset of the common methods described above in section 13.4.5 Management Class Methods on page 607:

Table 182 Performance Management Methods

Method Type	Value	Description
PerformanceGet()	0x01	Request a get (read) of a class specific information attribute
PerformanceSet()	0x02	Request a set (write) of a class specific information attribute.
PerformanceGetResp()	0x81	Response from a get or set request.

16.1.3 MANDATORY ATTRIBUTES

Performance Management defines the mandatory Attributes and Attribute Modifiers use summarized in Table 183 Mandatory Performance Management Attributes on page 750. They are described in detail in the sections following the table. The use model for these attributes is described in 16.1.3.6 Mandatory Performance Attribute Use Model on page 766.

Table 183 Mandatory Performance Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description	Length
ClassPortInfo	0x0001	0x00000000	See 13.4.8.1 ClassPortInfo on page 619	
PortSamplesControl	0x0010	Selects one of n independent sampling mechanisms; zero (0) must be implemented.	Port Performance Data Sampling Control	68 bytes
PortSamplesResult	0x0011	Selects one of n independent sampling mechanisms; zero (0) must be implemented.	Port Performance Data Sampling Results	64 bytes
PortCounters	0x0012	0x00000000	Port Basic Performance & Error Counters	40 bytes
Reserved	0x0013-0x0014	0x00000000-0xFFFFFFFF	Reserved	

Table 184 Mandatory Performance Management Attribute / Method Map

Attribute Name	PerformanceGet	PerformanceSet
ClassPortInfo	x	x
PortSamplesControl	x	x
PortSamplesResult	x	
PortCounters	x	x

16.1.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619.

Table 185 Performance Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619
8	AllPortSelect	If reported as 1, indicates that all attributes containing the PortSelect component support setting it to 0xFF to gather data from all ports at once. If reported as 0, using 0xFF in PortSelect results in undefined behavior.
9-15	-	Class-specific bits are reserved

16.1.3.2 PORTSAMPLESCONTROL

The PortSamplesControl attribute is mandatory. It provides a means of initiating a sample and selecting, for one selected port during the specified interval, quantities to be sampled such as:

- The amount of data sent and received
- The number of packets sent and received
- The transmit queue depth at the start of the interval

The complete list of quantities that can be sampled using this mechanism is given in Table 187 CounterSelect Values .

Sampling is initiated by means of a PerformanceSet(PortSamplesControl). Sampling status and results are obtained by means of a PerformanceGet(PortSamplesResult).

To support random sampling that is decoupled from MAD latencies and other port activities at either the sender or receiver, the PortSamplesControl attribute provides a means to specify a delayed start time for the sample interval. See the SampleStart component in Table 186 PortSamplesControl .

Performance sampling operations are based on a standard time interval called a tick. A tick is a multiple of the link transfer period. For example, a multiple of 400 picoseconds for a link running at 2.5 giga-transfers per second. Implementers are given a range of multipliers to choose from.

The Attribute Modifier selects one of several possible independent sampling mechanisms.

C16-3: All nodes shall implement PortSamplesControl and PortSamplesResult corresponding to an Attribute Modifier of zero. Implementation of additional sets of PortSamplesControl and PortSamplesResult permits simultaneous sampling of multiple ports, and shall use ascending Attribute Modifier values starting with one (1). The number of additional sets implemented is defined in PortSamplesControl.SampleMechanisms.

C16-4: For each sampling mechanisms, at least one and up to 15 counters shall be implemented.

Table 186 PortSamplesControl

Component	Access	Length (bits)	Description
OpCode	RW	8	Used to select a specific packet op code (as found in BTH) when sampling optional quantities that are op code specific. If OpCode is 0xFF, all op codes are sampled as one otherwise only one op code can be sampled at a time, although multiple quantities can be sampled for the same op code.
PortSelect	RW	8	Selects which port will be sampled. For an HCA or TCA, PortSelect refers to an endpoint. For a switch, PortSelect refers to a switch port. The valid values are 1 to the number of ports of the node; the management port of a switch is not included. If the value is invalid, the sample timers run normally but the resulting sample counter values are undefined. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause samples from all valid ports to be accumulated.
Tick	RO	8	Indicates the node's sampling clock interval as a multiple of 10x the link transfer period. For a 2.5 Gtransfer link, the transfer period is 400 picoseconds. The encoding is: 0x00 = 10 x link transfer period (4 nanoseconds for a 2.5 Gtransfer link) 0x01 = 20 x link transfer period 0x02 = 30 x link transfer period ... 0xFF = 2,560 x link transfer period To maximize utility of the performance attributes, implementers are encouraged to choose the smallest practical tick size
Reserved	RO	5	Reserved, shall be zero.

Table 186 PortSamplesControl

Component	Access	Length (bits)	Description
CounterWidth	RO	3	Indicates the actual width in bits of the following components: - SampleStart - SampleInterval - PortSamplesResult:Counter0 to 14 The encoding is: 0 = 16 bits 1 = 20 bits 2 = 24 bits 3 = 28 bits 4 = 32 bits 5 - 7 = reserved Counters smaller than 32 bits shall be implemented as the least significant bits of the corresponding 32-bit attribute component, with the unimplemented upper bits of the component returning zeroes for Get and ignored for Set.
Reserved	RO	2	Reserved, shall be zero.
Counter0Mask	RO	3	A bitmask that determines the capabilities of PortSamplesResult:Counter0. Bit 0 = supports all mandatory quantities; shall be 1 Bit 1 = supports optional quantities Bit 2 = supports vendor-defined quantities
CounterMasks1to9	RO	27	An array of nine 3-bit bitmasks, each of which determines the capabilities of an optional counter in PortSamplesResult. The most significant 3-bit field corresponds to PortSamplesResult:Counter1; the least significant field corresponds to PortSamplesResult:Counter9 Encoding: Bit 0 = supports all mandatory quantities Bit 1 = supports optional quantities Bit 2 = supports vendor-defined quantities All bits zero means the counter is not implemented
Reserved	RO	1	Reserved, shall be zero.
CounterMasks10to14	RO	15	An array of five 3-bit bitmasks, each of which determines the capabilities of an optional counter in PortSamplesResult. The most significant 3-bit field corresponds to PortSamplesResult:Counter10; the least significant field corresponds to PortSamplesResult:Counter14 Encoding: Bit 0 = supports all mandatory quantities Bit 1 = supports optional quantities Bit 2 = supports vendor-defined quantities All bits zero means the counter is not implemented

Table 186 PortSamplesControl

Component	Access	Length (bits)	Description
SampleMechanisms	RO	8	The number of independent sample mechanisms implemented (i.e. sets of PortSamplesControl and PortSamplesResult), minus one: 0 = one sample mechanism is available (addressed via Attribute Modifier zero) 1 = two sample mechanisms are available, Attribute Modifiers 0 and 1 ... 255 = 256 sample mechanisms are available, addressed via Attribute Modifiers 0 through 255 Providing multiple sampling mechanisms is optional. N sample mechanisms would permit N independent samples to be run simultaneously. A special value of the Attribute Modifier (0xFFFFFFFF) allows all sample mechanisms to be started with a single Set, sampling the same quantities during the same interval on N ports
Reserved	RO	6	Reserved, shall be zero.
SampleStatus	RO	2	Indicates the status of sampling: 0 = sampling is complete and the results are available from the PortSamplesResult attribute 1 = the SampleStart timer is running. All sample counter values in PortSamplesResult are undefined 2 = sampling is underway. All sample counter values in PortSamplesResult are undefined 3 = reserved While SampleStatus is non-zero, a PerformanceSet (PortSamplesControl) will not affect PortSamplesControl and will return the existing values of all components

Table 186 PortSamplesControl

Component	Access	Length (bits)	Description
OptionMask	RO	64	A bit mask indicating which optional InfiniBand performance quantities are implemented, if any. See Table 187 CounterSelect Values for a description of each quantity or set of quantities: Bit 0 (LSB) = reserved shall be zero. Bit 1 = PortXmitQueue[n] Bit 2 = PortXmitDataVL[n] Bit 3 = PortRcvDataVL[n] Bit 4 = PortXmitPktVL[n] Bit 5 = PortRcvPktVL[n] Bit 6 = PortRcvErrorDetails:PortLocalPhysicalErrors Bit 7 = PortRcvErrorDetails:PortMalformedPacketErrors Bit 8 = PortRcvErrorDetails: PortBufferOverrunErrors Bit 9 = PortRcvErrorDetails: PortDLIDMappingErrors Bit 10 = PortRcvErrorDetails: PortVLMappingErrors Bit 11 = PortRcvErrorDetails: PortLoopingErrors Bit 12 = PortXmitDiscardDetails: PortInactiveDiscards Bit 13 = PortXmitDiscardDetails: PortNeighborMTUDiscards Bit 14 = PortXmitDiscardDetails: PortSwLifetimeLimitDiscards Bit 15 = PortXmitDiscardDetails: PortSwHOQLifetimeLimitDiscards Bit 16 = PortOpRcvCounters: PortOpRcvPkts Bit 17 = PortOpRcvCounters: PortOpRcvData Bit 18 = PortFlowCtlCounters: PortXmitFlowPkts Bit 19 = PortFlowCtlCounters: PortRcvFlowPkts Bit 20 = PortVLOpPackets: PortVLOpPackets[n] Bit 21 = PortVLOpData: PortVLOpData[n] Bit 22 = PortVLXmitFlowCtlUpdateErrors: PortVLXmitFlowCtlUpdateErrors[n] Bit 23 = PortVLXmitWaitCounters: PortVLXmitWait[n] Bits 24 - 47 Reserved shall be zero. Bit 48 = SwPortVLUnkDests: PortVLUnkDests[n] Bits 49 -63 Reserved shall be zero. Performance quantities that are counted per VL are limited to the actual number of VLs implemented. The result of selecting an unimplemented quantity is all zeroes.
VendorMask	RO	64	A bitmask indicating which vendor-specific counters are implemented. Must be zero if the node does not support any vendor-specific counters, otherwise use is vendor-defined

Table 186 PortSamplesControl

Component	Access	Length (bits)	Description
SampleStart	RW	32	Determines when the sampling interval starts. When Set, this value is loaded into a timer and the following events occur: - SampleStatus is set to 1 - Counters in PortSamplesResult are set to zero - The timer begins decrementing once per tick. When the timer reaches zero, timing stops and the following events occur: - The PortXmitQueue quantities if selected are latched - PortSamplesResult counters are started - SampleStatus is set to 2 - The SampleInterval timer is started The SampleStart timer allows a performance application to randomize the sample start time and insure decoupling from node or network events. Values used will typically be 10's of milliseconds. It is the fine granularity of this interval with respect to the link rate that makes decoupling possible
SampleInterval	RW	32	Determines the length of the sampling interval. When Set, this value is loaded into a timer. When the SampleStart counter reaches zero, this timer begins decrementing once per tick. When it reaches zero, timing stops and the following events occur: - PortSamplesResult counters are stopped and the resulting values made available - SampleStatus is set to zero
Tag	RW	16	Used by a performance application when it does a PerformanceSet (PortSamplesControl) to uniquely identify its sample run in case of a collision with another performance application When an application wishes to start a sample run, it should pick a random Tag value and do a PerformanceSet (PortSamplesControl). If the returned value of Tag does not match the selected value, another application is using the sampling mechanism. In this case the first application must wait for a suitable time and retry its sample
CounterSelect0	RW	16	Selects quantity to be sampled by PortSamplesResult:Counter0 as defined in Table 187 CounterSelect Values on page 758. If an unimplemented quantity is selected, a Get to PortSamplesResult:Counter0 returns zeroes
CounterSelect1	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter1
CounterSelect2	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter2
CounterSelect3	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter3
CounterSelect4	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter4
CounterSelect5	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter5

Table 186 PortSamplesControl

Component	Access	Length (bits)	Description
CounterSelect6	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter6
CounterSelect7	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter7
CounterSelect8	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter8
CounterSelect9	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter9
CounterSelect10	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter10
CounterSelect11	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter11
CounterSelect12	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter12
CounterSelect13	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter13
CounterSelect14	RW	16	Similar to CounterSelect0; selects quantity to be sampled by PortSamplesResult:Counter14

16.1.3.3 COUNTERSELECT VALUES

Table 187 CounterSelect Values on page 758 lists the values that can be used in the CounterSelect[n] components of the PortSamplesControl attribute to select a particular quantity to sample.

Quantities that can be sampled are divided into 3 ranges:

- Mandatory quantities (0x0000 - 0x3FFF).

C16-5: Mandatory quantities for performance sampling shall be implemented on all ports of all nodes.

- Optional quantities (0x4000 - 0xBFFF).

o16-1: If provided, optional quantities for performance sampling shall be implemented as described.

- Vendor quantities (0xC000 - 0xFFFF). Vendors may define and implement their own quantities in this range

Table 187 CounterSelect Values

Sample Select Value	Name	Description
Mandatory Quantities		
0x0000	Reserved	Reserved
0x0001	PortXmitData	Total number of data octets, divided by 4, transmitted on all VLs during the sampling interval from the port selected by PortSelect. This includes all octets between (and not including) the start of packet delimiter and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported in units of four octets.
0x0002	PortRcvData	Total number of data octets, divided by 4, received on all VLs during the sampling interval on the port selected by PortSelect. This includes all octets between (and not including) the start of packet delimiter and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported in units of four octets.
0x0003	PortXmitPkts	Total number of packets, excluding link packets, transmitted on all VLs during the sampling interval from the port selected by PortSelect.
0x0004	PortRcvPkts	Total number of packets, including packets containing errors and excluding link packets, received on all VLs during the sampling interval on the port selected by PortSelect.
0x0005	PortXmitWait	The number of ticks during which the port selected by PortSelect had data to transmit but no data was sent during the entire tick either because of insufficient credits or because of lack of arbitration.
0x0007-0x3FFF	Reserved	Reserved. Result of sampling is all zeroes.

Table 187 CounterSelect Values

Sample Select Value	Name	Description
Optional InfiniBand Quantities All quantities that are available in optional attributes as running counters are also optionally available for sampling over a given period. Each sampling counter corresponding to an optional running counter is reset to zero for each sample and increments along with the selected running counter during the sampling interval. For certain quantities, such as PortXmitQueue[n], there are no corresponding optional attributes. All values between 0x4000 and 0xBFFF not listed here are reserved and the result of sampling is all zeroes		
0x4n00	PortXmitQueue[n]	Contains the transmit queue depth in bytes on VL “n” of the port selected by PortSelect at the time the SampleStart timer expired The goal of measuring queue depths is to enable software to compute the average time data waits for transmission inside a node. Ideally, a node should increment a counter upon arrival of each byte that is destined for a given output port and should decrement the counter upon departure of each byte from the output port. In practice, this will be impossible to implement precisely. Implementers are encouraged to measure queue depths as accurately as practical and to document any systematic measurement errors. Note that an implementation can compensate for an inherent delay in accounting for arriving bytes by introducing an equal delay in accounting for departing bytes
0x4n01	PortXmitDataVL[n]	Total number of data octets, divided by 4, transmitted on VL “n” from the port selected by PortSelect. This includes all octets between the start of packet and end of packet delimiters. It excludes all control groups and VCRCs. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets
0x4n02	PortRcvDataVL[n]	Total number of data octets, divided by 4, received on input VL “n” on the port selected by PortSelect. This includes all octets between the start of packet and end of packet delimiters. It excludes all control groups and VCRCs. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets
0x4n03	PortXmitPktVL[n]	Total number of packets transmitted on VL “n” from the port selected by PortSelect with or without errors.
0x4n04	PortRcvPktVL[n]	Total number of packets received on input VL “n” from the port selected by PortSelect with or without errors.

Table 187 CounterSelect Values

Sample Select Value	Name	Description
0x4005	PortRcvErrorDetails: PortLocalPhysicalErrors	See Table 192 PortRcvErrorDetails on page 769.
0x4006	PortRcvErrorDetails: PortMalformedPacket-Errors	See Table 192 PortRcvErrorDetails on page 769.
0x4007	PortRcvErrorDetails: PortBufferOverrunEr-rors	See Table 192 PortRcvErrorDetails on page 769.
0x4008	PortRcvErrorDetails: PortDLIDMappingErrors	See Table 192 PortRcvErrorDetails on page 769.
0x4009	PortRcvErrorDetails: PortVLMappingErrors	See Table 192 PortRcvErrorDetails on page 769.
0x400A	PortRcvErrorDetails: PortLoopingErrors	See Table 192 PortRcvErrorDetails on page 769.
0x400B	PortXmitDiscardDe-tails: PortInactiveDis-cards	See Table 193 PortXmitDiscardDetails on page 770.
0x400C	PortXmitDiscardDe-tails: PortNeighborM-TUDiscards	See Table 193 PortXmitDiscardDetails on page 770.
0x400D	PortXmitDiscardDe-tails: PortSwLifetime-LimitDiscards	See Table 193 PortXmitDiscardDetails on page 770.
0x400E	PortXmitDiscardDe-tails: PortSwHOQLimit-Discards	See Table 193 PortXmitDiscardDetails on page 770.
0x400F	PortOpRcvCounters: PortOpRcvPkts	See Table 194 PortOpRcvCounters on page 770. The op code to be sampled is selected by PortSamples-Control: OpCode.
0x4010	PortOpRcvCounters: PortOpRcvData	See Table 194 PortOpRcvCounters on page 770. The op code to be sampled is selected by PortSamples-Control: OpCode.
0x4011	PortFlowCtlCounters: PortXmitFlowPkts	See Table 195 PortFlowCtlCounters on page 771.
0x4012	PortFlowCtlCounters: PortRcvFlowPkts	See Table 195 PortFlowCtlCounters on page 771
0x4n13	PortVLOpPackets: PortVLOpPackets[n]	See Table 196 PortVLOpPackets on page 772. The op code to be sampled is selected by PortSamples-Control: OpCode

Table 187 CounterSelect Values

Sample Select Value	Name	Description
0x4n14	PortVLOpData: PortVLOpData[n]	See Table 197 PortVLOpData on page 774. The op code to be sampled is selected by PortSamplesControl: OpCode
0x4n15	PortVLXmitFlowCtlUpdateErrors: PortVLXmitFlowCtlUpdateErrors[n]	See Table 198 PortVLXmitFlowCtlUpdateErrors on page 775.
0x4n16	PortVLXmitWaitCounters: PortVLXmitWait[n]	See Table 199 PortVLXmitWaitCounters on page 777
0x4n30	SwPortVLUnkDests: PortVLUnkDests[n]	See Table 200 SwPortVLCongestion on page 779
	Vendor-Defined Quantities	
0xC000-0xFFFF	Reserved	Reserved for vendor-specific counters

16.1.3.4 PORTSAMPLESRESULT

This mandatory attribute reports the results of a particular sample controlled and initiated via the PortSamplesControl attribute.

Table 188 PortSamplesResult

Component	Access	Length (bits)	Description
Tag	RO	16	Read-only copy of PortSamplesControl:Tag. The Tag mechanism provides a means for performance applications to detect collisions when using the sampling mechanism. After successfully initiating a sample run, an application should wait until the sample should have completed, then repeat a PerformanceGet (PortSamplesResult) until SampleStatus is zero. If after any Get the Tag value in the result does not match the value set by the application at the start of the run, another application has already started a new sample. In this case the first application must wait for a suitable time and retry its sample
Reserved	RO	14	Reserved, shall be zero.
SampleStatus	RO	2	Read-only copy of PortSamplesControl:SampleStatus. Provided here to minimize traffic while application is polling for sample completion

Table 188 PortSamplesResult

Component	Access	Length (bits)	Description
Counter0	RO	32	Mandatory counter. When PortSamplesControl:SampleStatus is zero, contains the result of sampling the quantity selected by PortSamplesControl:CounterSelect0. Undefined when PortSamplesControl:SampleStatus is non-zero. The actual number of valid (least significant) bits in the counter is defined by PortSamplesControl:CounterWidth
Counter1	RO	32	Optional counter. All zeroes if not implemented; otherwise similar to Counter0. Contains the result of sampling the quantity selected by PortSamplesControl:Counter1Select
Counter2	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter2Select
Counter3	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter3Select
Counter4	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter4Select
Counter5	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter5Select
Counter6	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter6Select
Counter7	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter7Select
Counter8	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter8Select
Counter9	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter9Select
Counter10	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter10Select
Counter11	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter11Select

Table 188 PortSamplesResult

Component	Access	Length (bits)	Description
Counter12	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter12Select
Counter13	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter13Select
Counter14	RO	32	Similar to Counter1; contains the result of sampling the quantity selected by PortSamplesControl:Counter14Select

16.1.3.5 PORTCOUNTERS

C16-6: The PortCounters attribute of the Performance class is mandatory.

It provides basic performance and exception statistics for a port.

C16-7: When initially powered-up or reset, the value of all counters on all ports of a node shall be set to zero. During operation, instead of overflowing, they shall stop at all ones. At any time, writing (Set) zero into a counter shall cause the counter to be reset to zero.

Note that writing (Set) anything other than zero into a counter results in undefined behavior.

Note that although PortCounters is mandatory, it contains components that are optional.

Table 189 PortCounters

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.

Table 189 PortCounters

Component	Access	Length (bits)	Description
PortSelect	RW	8	When reading (Get), selects the port for which the data is reported. The valid values are 1 to the number of ports of the node; the management port of a switch is not included. If the value is invalid, the counter data returned is undefined. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause PerformanceSet(PortCounters) to alter the content of the selected counters (by CounterSelect) for all the valid ports and a PerformanceGetResp(PortCounters) returned in response to a PerformanceGet(PortCounters) or a PerformanceSet(PortCounters) to fill each counter component value with the sum of the respective counter values for all the valid ports.
CounterSelect	RW	16	When writing (Set), selects which counters are affected by the operation. When reading (Get), this is ignored. Bit 0 - SymbolErrorCounter Bit 1 - LinkErrorRecoveryCounter Bit 2 - LinkDownedCounter Bit 3 - PortRcvErrors Bit 4 - PortRcvRemotePhysicalErrors Bit 5 - PortRcvSwitchRelayErrors Bit 6 - PortXmitDiscards Bit 7 - PortXmitConstraintErrors Bit 8 - PortRcvConstraintErrors Bit 9 - LocalLinkIntegrityErrors Bit 10 - ExcessiveBufferOverrunErrors Bit 11 - VL15Dropped Bit 12 - PortXmitData Bit 13 - PortRcvData Bit 14 - PortXmitPkts Bit 15 - PortRcvPkts
SymbolErrorCounter	RW	16	Total number of symbol errors detected on one or more lanes. Refer to Volume 2.
LinkErrorRecovery-Counter	RW	8	Total number of times the Port Training state machine has successfully completed the link error recovery process. Refer to Volume 2.
LinkDownedCounter	RW	8	Total number of times the Port Training state machine has failed the link error recovery process and downed the link. Refer to Volume 2.

Table 189 PortCounters

Component	Access	Length (bits)	Description
PortRcvErrors	RW	16	Total number of packets containing an error that were received on the port. These errors include: - Local physical errors (ICRC, VCRC, FCCRC, and all physical errors that cause entry into the BAD PACKET or BAD PACKET DISCARD states of the packet receiver state machine) - Malformed data packet errors (LVer, length, VL) - Malformed link packet errors (operand, length, VL) - Packets discarded due to buffer overrun
PortRcvRemotePhysicalErrors	RW	16	Total number of packets marked with the EBP delimiter received on the port.
PortRcvSwitchRelayErrors	RW	16	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay. Reasons for this include: - DLID mapping - VL mapping - looping (output port = input port)
PortXmitDiscards	RW	16	Total number of outbound packets discarded by the port because the port is down or congested. Reasons for this include: - output port is in the inactive state - packet length exceeded neighbor MTU - switch lifetime limit exceeded - switch HOQ limit exceeded
PortXmitConstraintErrors	RW	8	Total number of packets not transmitted from the port for the following reasons: - FilterRawOutbound is true and packet is raw - PartitionEnforcementOutbound is true and packet fails partition key check, IP version check, or transport header version check
PortRcvConstraintErrors	RW	8	Total number of packets received on the port that are discarded for the following reasons: - FilterRawInbound is true and packet is raw - PartitionEnforcementInbound is true and packet fails partition key check, IP version check, or transport header version check
Reserved1	RO	8	Reserved
LocalLinkIntegrityErrors	RW	4	The number of times that the frequency of packets containing local physical errors exceeded local_phy_errors, see Table 126 PortInfo on page 665.

Table 189 PortCounters

Component	Access	Length (bits)	Description
ExcessiveBufferOver-runErrors	RW	4	The number of times that overrun_errors consecutive flow control update periods occurred with at least one overrun error in each period, see Table 126 PortInfo on page 665.
Reserved2	RO	16	Reserved
VL15Dropped	RW	16	Number of incoming VL15 packets dropped due to resource limitations on port selected by PortSelect (due to lack of buffers)
PortXmitData	RW	32	Optional; shall be zero if not implemented. Total number of data octets, divided by 4, transmitted on all VLs from the port selected by PortSelect. This includes all octets between (and not including) the start of packet delimiter and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets.
PortRcvData	RW	32	Optional; shall be zero if not implemented. Total number of data octets, divided by 4, received on all VLs on the port selected by PortSelect. This includes all octets between (and not including) the start of packet delimiter and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets.
PortXmitPkts	RW	32	Optional; shall be zero if not implemented. Total number of packets, excluding link packets, transmitted on all VLs from the port.
PortRcvPkts	RW	32	Optional; shall be zero if not implemented. Total number of packets, including packets containing errors and excluding link packets, received from all VLs on the port.

16.1.3.6 MANDATORY PERFORMANCE ATTRIBUTE USE MODEL

The PortCounters information is used in the standard manner using PerformanceGet() to read it and PerformanceSet() to reset the counters.

PortSamplesControl and PortSamplesResult are used together to sample one or more quantities over a specified period of time:

The application can first determine the node's sampling capabilities via a PerformanceGet(PortSamplesControl). This will return the number and width of available counters, the quantities that can be sampled, and the basic time interval (tick). From these the application can compute the maximum sample interval that will not cause counter overflow.

C16-8: To initiate a sample, the application shall do the following:

- Select a random value for SampleStart. The SampleStart timer allows a performance application to randomize the sample start time and insure decoupling from node or network events. Values used will typically be 10’s of milliseconds.
- Select a random Tag value. This value is used to detect collisions among multiple independent performance applications accessing the same node
- Select a SampleInterval value, the quantities to be sampled, and the counter that will be assigned to count each quantity.
- Do a PerformanceSet (PortSamplesControl). If the returned value of Tag does not match the selected value, another application is using the sampling mechanism. In this case the first application must wait for a suitable time and retry the PerformanceSet().
- Once the sample has been successfully started, the application should wait until the SampleStart and SampleInterval timers should have expired, then repeat PerformanceGet (PortSamplesResult) until SampleStatus is zero. If at any time the returned Tag value no longer matches the application’s chosen value, regardless of SampleStatus, it means another application has gained control of the sampling mechanism. In this case the first application must restart the sampling process.

If more than one set of sampling mechanisms is implemented, the additional ones are addressed using non-zero Attribute Modifier values. The previous use model applies to each pair of PortSamplesControl and PortSamplesResult, treating each as an independent entity.

16.1.4 OPTIONAL ATTRIBUTES

Performance Management defines the optional Attributes and Attribute Modifier use summarized in Table 190 Optional Performance Management Attributes . They are described in detail in the sections following the table. All quantities in these attributes can also be sampled via the PortSamplesControl mechanism. The optional Attributes available are reflected by the OptionMask in the PortSamplesControl attribute.

o16-2: All counters within these optional Performance Attributes shall be initialized to zero; instead of overflowing they shall stop at all ones and shall be reset by a management application.

Table 190 Optional Performance Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description	Length
PortRcvErrorDetails	0x15	0x00000000	Port Detailed Error Counters	16 bytes
PortXmitDiscardDetails	0x16	0x00000000	Port Transmit Discard Counters	12 bytes
PortOpRcvCounters	0x17	0x00000000	Port Receive Counters per Op Code	12 bytes
PortFlowCtlCounters	0x18	0x00000000	Port Flow Control Counters	12 bytes
PortVLOpPackets	0x19	0x00000000	Port Packets Received per Op Code per VL	36 bytes
PortVLOpData	0x1A	0x00000000	Port Kilobytes Received per Op Code per VL	68 bytes
PortVLXmitFlowCtlUpdateErrors	0x1B	0x00000000	Port Flow Control update errors per VL	8 bytes
PortVLXmitWaitCounters	0x1C	0x00000000	Port Ticks Waiting to Transmit Counters per VL	36 bytes
SwPortVLCongestion	0x30	0x00000000	Switch Port Congestion per VL	36 bytes

Table 191 Optional Performance Management Attribute / Method Map

Attribute Name	PerformanceGet	PerformanceSet
PortRcvErrorDetails	x	x
PortXmitDiscardDetails	x	x
PortOpRcvCounters	x	x
PortFlowCtlCounters	x	x
PortVLOpPackets	x	x
PortVLOpData	x	x
PortVLXmitFlowCtlUpdateErrors	x	x
PortVLXmitWaitCounters	x	x
SwPortVLCongestion	x	x

16.1.4.1 PORTRCVERRORDETAILS

Table 192 PortRcvErrorDetails

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.
PortSelect	RW	8	Selects the port for which the statistics are reported. Statistics are accumulated for all VLs on a port. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.
CounterSelect	RW	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortLocalPhysicalErrors Bit 1 - PortMalformedPacketErrors Bit 2 - PortBufferOverrunErrors Bit 3 - PortDLIDMappingErrors Bit 4 - PortVLMappingErrors Bit 5 - PortLoopingErrors Bits 6 to 15 - Reserved
PortLocalPhysicalErrors	RW	16	Total number of packets received on the port that contain local physical errors (ICRC, VCRC, FCCRC, and all physical errors that cause entry into the BAD PACKET or BAD PACKET DISCARD states of the packet receiver state machine).
PortMalformedPacketErrors	RW	16	Total number of packets received on the port that contain malformed packet errors - data packets: LVer, length, VL - link packets: operand, length, VL
PortBufferOverrunErrors	RW	16	Total number of packets received on the port that were discarded due to buffer overrun.
PortDLIDMappingErrors	RW	16	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay due to DLID mapping errors.
PortVLMappingErrors	RW	16	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay due to VL mapping errors.
PortLoopingErrors	RW	16	Total number of packets received on the port that were discarded because they could not be forwarded by the switch relay due to looping errors (output port = input port).

16.1.4.2 PORTXMITDISCARDDETAILS

Table 193 PortXmitDiscardDetails

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.
PortSelect	RW	8	Selects the port for which the statistics are reported. Statistics are accumulated for all VLs on a port. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.
CounterSelect	RW	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortInactiveDiscards Bit 1 - PortNeighborMTUDiscards Bit 2 - PortSwLifetimeLimitDiscards Bit 3 - PortSwHOQLimitDiscards Bits 4 to 15 - Reserved
PortInactiveDiscards	RW	16	Total number of outbound packets discarded by the port because it is in the inactive state.
PortNeighborMTUDiscards	RW	16	Total number of outbound packets discarded by the port because packet length exceeded the neighbor MTU.
PortSwLifetimeLimitDiscards	RW	16	Total number of outbound packets discarded by the port because the switch lifetime limit was exceeded. Applies to switches only.
PortSwHOQLimitDiscards	RW	16	Total number of outbound packets discarded by the port because the switch HOQ lifetime was exceeded. Applies to switches only.

16.1.4.3 PORTOPRCVCOUNTERS

Table 194 PortOpRcvCounters

Component	Access	Length (bits)	Description
OpCode	RW	8	Selects the op code (as found in BTH) for which the statistics are reported. 0xFF means all op codes.

Table 194 PortOpRcvCounters

Component	Access	Length (bits)	Description
PortSelect	RW	8	Selects the port for which the statistics are reported. Statistics are accumulated for all VLs on a port. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.
CounterSelect	RW	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortOpRcvPkts Bit 1 - PortOpRcvData Bits 2 to 15 - Reserved
PortOpRcvPkts	RW	32	Total number of packets received without error on the port selected by PortSelect containing the opcode selected by OpCode.
PortOpRcvData	RW	32	Total number of data octets, divided by 4, received without error on all VLs from the port selected by PortSelect containing the opcode selected by OpCode. This includes all octets between (and not including) the start of packet delimiter and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets.

16.1.4.4 PORTFLOWCTLCOUNTERS

Table 195 PortFlowCtlCounters

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.
PortSelect	RW	8	Selects the port for which the statistics are reported. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.

Table 195 PortFlowCtlCounters

Component	Access	Length (bits)	Description
CounterSelect	RW	16	When writing (Set), selects which counters are over-written by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortXmitFlowPkts Bit 1 - PortRcvFlowPkts Bits 2 to 15 - Reserved
PortXmitFlow-Pkts	RW	32	Total number of flow control packets transmitted on the port selected by PortSelect
PortRcvFlowP-kts	RW	32	Total number of flow control packets received on the port selected by PortSelect

16.1.4.5 PORTVLOPPACKETS

Table 196 PortVLOpPackets

Component	Access	Length (bits)	Description
OpCode	RW	8	Selects the op code (as found in BTH) for which the statistics are reported. 0xFF means all op codes.
PortSelect	RW	8	Selects the port for which the statistics are reported. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPort-Info:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumu-lated.

Table 196 PortVLOpPackets

Component	Access	Length (bits)	Description
CounterSelect	RW	16	When writing (Set), selects which counters are over-written by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLOpPackets0 Bit 1 - PortVLOpPackets1 Bit 2 - PortVLOpPackets2 Bit 3 - PortVLOpPackets3 Bit 4 - PortVLOpPackets4 Bit 5 - PortVLOpPackets5 Bit 6 - PortVLOpPackets6 Bit 7 - PortVLOpPackets7 Bit 8 - PortVLOpPackets8 Bit 9 - PortVLOpPackets9 Bit 10 - PortVLOpPackets10 Bit 11 - PortVLOpPackets11 Bit 12 - PortVLOpPackets12 Bit 13 - PortVLOpPackets13 Bit 14 - PortVLOpPackets14 Bit 15 - PortVLOpPackets15
PortVLOpPackets0	RW	16	The total number of packets received without error on VL 0 of the port selected by PortSelect containing the opcode selected by OpCode
PortVLOpPackets1	RW	16	Similar count for VL 1
PortVLOpPackets2	RW	16	Similar count for VL 2
PortVLOpPackets3	RW	16	Similar count for VL 3
PortVLOpPackets4	RW	16	Similar count for VL 4
PortVLOpPackets5	RW	16	Similar count for VL 5
PortVLOpPackets6	RW	16	Similar count for VL 6
PortVLOpPackets7	RW	16	Similar count for VL 7
PortVLOpPackets8	RW	16	Similar count for VL 8
PortVLOpPackets9	RW	16	Similar count for VL 9
PortVLOpPackets10	RW	16	Similar count for VL 10
PortVLOpPackets11	RW	16	Similar count for VL 11
PortVLOpPackets12	RW	16	Similar count for VL 12
PortVLOpPackets13	RW	16	Similar count for VL 13
PortVLOpPackets14	RW	16	Similar count for VL 14
PortVLOpPackets15	RW	16	Similar count for VL 15

16.1.4.6 PORTVLOPDATA

Table 197 PortVLOpData

Component	Access	Length (bits)	Description
OpCode	RW	8	Selects the op code (as found in BTH) for which the statistics are reported. 0xFF means all op codes.
PortSelect	RW	8	Selects the port for which the statistics are reported. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.
CounterSelect	RW	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLOpData0 Bit 1 - PortVLOpData1 Bit 2 - PortVLOpData2 Bit 3 - PortVLOpData3 Bit 4 - PortVLOpData4 Bit 5 - PortVLOpData5 Bit 6 - PortVLOpData6 Bit 7 - PortVLOpData7 Bit 8 - PortVLOpData8 Bit 9 - PortVLOpData9 Bit 10 - PortVLOpData10 Bit 11 - PortVLOpData11 Bit 12 - PortVLOpData12 Bit 13 - PortVLOpData13 Bit 14 - PortVLOpData14 Bit 15 - PortVLOpData15
PortVLOpData0	RW	32	Total number of data octets, divided by 4, received without error on VL 0 from the port selected by PortSelect containing the opcode selected by OpCode. This includes all octets between (and not including) the start of packet and VCRC. It excludes all link packets. Implementers may choose to count data octets in groups larger than four but are encouraged to choose the smallest group possible. Results are still reported as a multiple of four octets
PortVLOpData1	RW	32	Similar count for VL 1
PortVLOpData2	RW	32	Similar count for VL 2

Table 197 PortVLOpData

Component	Access	Length (bits)	Description
PortVLOpData3	RW	32	Similar count for VL 3
PortVLOpData4	RW	32	Similar count for VL 4
PortVLOpData5	RW	32	Similar count for VL 5
PortVLOpData6	RW	32	Similar count for VL 6
PortVLOpData7	RW	32	Similar count for VL 7
PortVLOpData8	RW	32	Similar count for VL 8
PortVLOpData9	RW	32	Similar count for VL 9
PortVLOpData10	RW	32	Similar count for VL 10
PortVLOpData11	RW	32	Similar count for VL 11
PortVLOpData12	RW	32	Similar count for VL 12
PortVLOpData13	RW	32	Similar count for VL 13
PortVLOpData14	RW	32	Similar count for VL 14
PortVLOpData15	RW	32	Similar count for VL 15

16.1.4.7 PORTVLXMITFLOWCTLUPDATEERRORS

Table 198 PortVLXmitFlowCtlUpdateErrors

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.
PortSelect	RW	8	Selects the port for which the statistics are reported. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.

Table 198 PortVLXmitFlowCtlUpdateErrors

Component	Access	Length (bits)	Description
CounterSelect	RW	16	When writing (Set), selects which counters are overwritten by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLXmitFlowCtlUpdateErrors0 Bit 1 - PortVLXmitFlowCtlUpdateErrors1 Bit 2 - PortVLXmitFlowCtlUpdateErrors2 Bit 3 - PortVLXmitFlowCtlUpdateErrors3 Bit 4 - PortVLXmitFlowCtlUpdateErrors4 Bit 5 - PortVLXmitFlowCtlUpdateErrors5 Bit 6 - PortVLXmitFlowCtlUpdateErrors6 Bit 7 - PortVLXmitFlowCtlUpdateErrors7 Bit 8 - PortVLXmitFlowCtlUpdateErrors8 Bit 9 - PortVLXmitFlowCtlUpdateErrors9 Bit 10 - PortVLXmitFlowCtlUpdateErrors10 Bit 11 - PortVLXmitFlowCtlUpdateErrors11 Bit 12 - PortVLXmitFlowCtlUpdateErrors12 Bit 13 - PortVLXmitFlowCtlUpdateErrors13 Bit 14 - PortVLXmitFlowCtlUpdateErrors14 Bit 15 - PortVLXmitFlowCtlUpdateErrors15
PortVLXmit-FlowCtlUpdate Errors0	RW	2	Total number of flow control update errors on VL 0 on the port selected by PortSelect
PortVLXmit-FlowCtlUpdate Errors1	RW	2	Similar count for VL 1
PortVLXmit-FlowCtlUpdate Errors2	RW	2	Similar count for VL 2
PortVLXmit-FlowCtlUpdate Errors3	RW	2	Similar count for VL 3
PortVLXmit-FlowCtlUpdate Errors4	RW	2	Similar count for VL 4
PortVLXmit-FlowCtlUpdate Errors5	RW	2	Similar count for VL 5
PortVLXmit-FlowCtlUpdate Errors6	RW	2	Similar count for VL 6
PortVLXmit-FlowCtlUpdate Errors7	RW	2	Similar count for VL 7

Table 198 PortVLXmitFlowCtlUpdateErrors

Component	Access	Length (bits)	Description
PortVLXmit-FlowCtlUpdate Errors8	RW	2	Similar count for VL 8
PortVLXmit-FlowCtlUpdate Errors9	RW	2	Similar count for VL 9
PortVLXmit-FlowCtlUpdate Errors10	RW	2	Similar count for VL 10
PortVLXmit-FlowCtlUpdate Errors11	RW	2	Similar count for VL 11
PortVLXmit-FlowCtlUpdate Errors12	RW	2	Similar count for VL 12
PortVLXmit-FlowCtlUpdate Errors13	RW	2	Similar count for VL 13
PortVLXmit-FlowCtlUpdate Errors14	RW	2	Similar count for VL 14
PortVLXmit-FlowCtlUpdate Errors15	RW	2	Similar count for VL 15

16.1.4.8 PORTVLXMITWAITCOUNTERS

Table 199 PortVLXmitWaitCounters

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.
PortSelect	RW	8	Selects the port for which the statistics are reported. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.

Table 199 PortVLXmitWaitCounters

Component	Access	Length (bits)	Description
CounterSelect	RW	16	When writing (Set), selects which counters are over-written by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - PortVLXmitWait0 Bit 1 - PortVLXmitWait1 Bit 2 - PortVLXmitWait2 Bit 3 - PortVLXmitWait3 Bit 4 - PortVLXmitWait4 Bit 5 - PortVLXmitWait5 Bit 6 - PortVLXmitWait6 Bit 7 - PortVLXmitWait7 Bit 8 - PortVLXmitWait8 Bit 9 - PortVLXmitWait9 Bit 10 - PortVLXmitWait10 Bit 11 - PortVLXmitWait11 Bit 12 - PortVLXmitWait12 Bit 13 - PortVLXmitWait13 Bit 14 - PortVLXmitWait14 Bit 15 - PortVLXmitWait15
PortVLXmitWait0	RW	16	Total number of ticks during which the port selected by PortSelect had data to transmit on VL0 but no data was sent during the entire tick either because of insufficient credits or because of lack of arbitration.
PortVLXmitWait1	RW	16	Similar count for VL 1
PortVLXmitWait2	RW	16	Similar count for VL 2
PortVLXmitWait3	RW	16	Similar count for VL 3
PortVLXmitWait4	RW	16	Similar count for VL 4
PortVLXmitWait5	RW	16	Similar count for VL 5
PortVLXmitWait6	RW	16	Similar count for VL 6
PortVLXmitWait7	RW	16	Similar count for VL 7
PortVLXmitWait8	RW	16	Similar count for VL 8
PortVLXmitWait9	RW	16	Similar count for VL 9

Table 199 PortVLXmitWaitCounters

Component	Access	Length (bits)	Description
PortVLXmitWait10	RW	16	Similar count for VL 10
PortVLXmitWait11	RW	16	Similar count for VL 11
PortVLXmitWait12	RW	16	Similar count for VL 12
PortVLXmitWait13	RW	16	Similar count for VL 13
PortVLXmitWait14	RW	16	Similar count for VL 14
PortVLXmitWait15	RW	16	Similar count for VL 15

16.1.4.9 SwPortVLCongestion

Unlike the rest of the performance attributes described in this chapter, which apply to all node types, this optional attribute applies only to Switches.

Table 200 SwPortVLCongestion

Component	Access	Length (bits)	Description
Reserved	RO	8	Reserved, shall be zero.
PortSelect	RW	8	Selects the port for which the statistics are reported. Selecting a non-existent port results in all zeroes. If gathering data from all ports at once is supported (see Table 185 Performance Management ClassPortInfo:CapabilityMask on page 751), setting PortSelect to 0xFF will cause data from all ports to be accumulated.

Table 200 SwPortVLCongestion

Component	Access	Length (bits)	Description
CounterSelect	RW	16	When writing (Set), selects which counters are over-written by the values specified in their respective fields. When reading (Get), this is ignored. Bit 0 - SWPortVLCongestion0 Bit 1 - SWPortVLCongestion1 Bit 2 - SWPortVLCongestion2 Bit 3 - SWPortVLCongestion3 Bit 4 - SWPortVLCongestion4 Bit 5 - SWPortVLCongestion5 Bit 6 - SWPortVLCongestion6 Bit 7 - SWPortVLCongestion7 Bit 8 - SWPortVLCongestion8 Bit 9 - SWPortVLCongestion9 Bit 10 - SWPortVLCongestion10 Bit 11 - SWPortVLCongestion11 Bit 12 - SWPortVLCongestion12 Bit 13 - SWPortVLCongestion13 Bit 14 - SWPortVLCongestion14 Bit 15 - SWPortVLCongestion15
SWPortVLCongestion0	RW	16	Total number of packets to be transmitted on VL 0 of the output port selected by PortSelect that were discarded because of congestion. This includes the following reasons: - Switch lifetime limit exceeded - Switch HOQ limit exceeded
SWPortVLCongestion1	RW	16	Similar count for VL 1.
SWPortVLCongestion2	RW	16	Similar count for VL 2.
SWPortVLCongestion3	RW	16	Similar count for VL 3.
SWPortVLCongestion4	RW	16	Similar count for VL 4.
SWPortVLCongestion5	RW	16	Similar count for VL 5.
SWPortVLCongestion6	RW	16	Similar count for VL 6
SWPortVLCongestion7	RW	16	Similar count for VL 7
SWPortVLCongestion8	RW	16	Similar count for VL 8
SWPortVLCongestion9	RW	16	Similar count for VL 9
SWPortVLCongestion10	RW	16	Similar count for VL 10
SWPortVLCongestion11	RW	16	Similar count for VL 11

Table 200 SwPortVLCongestion

Component	Access	Length (bits)	Description
SWPortVLCongestion12	RW	16	Similar count for VL 12
SWPortVLCongestion13	RW	16	Similar count for VL 13
SWPortVLCongestion14	RW	16	Similar count for VL 14
SWPortVLCongestion15	RW	16	Similar count for VL 15

16.2 BASEBOARD MANAGEMENT

C16-9: The Baseboard Management Agent is mandatory on all nodes.

This section describes the Management Datagrams used to transport Baseboard Management commands across the fabric. For more information regarding Hardware Management of IB Modules, non-Modules (IB devices whose packaging are different from an IB Module form factor), and Chassis, see InfiniBand Architecture Specification, Volume 2, Chapter “Hardware Management”. A simplified overview is presented here.

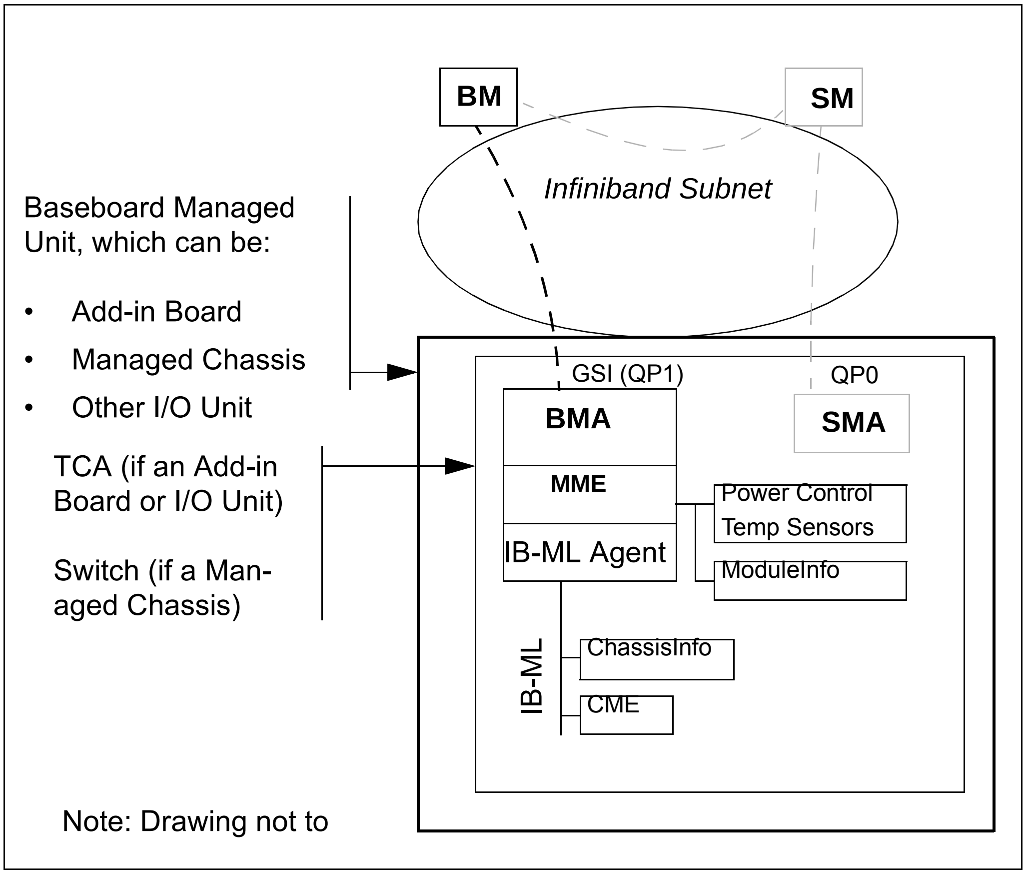


Figure 174 Baseboard Management Architecture

The SM and SMA are shown in Figure 174 strictly to indicate that Baseboard Management and Subnet Management are independent, separate entities in the fabric providing non-overlapping functionality.

The Baseboard Manager (BM) is a software entity that manages the hardware via Baseboard Management messages. From the BM, these messages are tunneled (encapsulated in MADs) through the IBA fabric to the Baseboard Management Agent (BMA), which then recognizes the message and forwards it to the Module Management Entity (MME). The MME processes the embedded Baseboard Management commands. In some cases, this results in the MME generating corresponding messages and transactions on a present InfiniBand Management Link (IB-ML). IB-ML messages may interface with a present Chassis Management Entity (CME).

The BM may use the Subnet Administration Interface to retrieve information regarding the nodes discovered in the IBA subnet, such as addresses, capabilities, types.

The Subnet Administration Interface provides basic discovery information that is common to all IBA endnodes, regardless of the type of Endnode in the subnet as described above. It does not provide information beyond this, such as VPD, chassis management data, and any other information under Baseboard Management control. This information is “discovered” through baseboard management.

A Baseboard Managed Unit can be either an IB-Module as defined in Volume 2, a form factor other than defined in Volume 2 (a non-Module), or a Managed Chassis. Protocol-aware IB-Modules handle the send/receive of the Baseboard Management MADs. The MADs are addressed using the LID of any endport of the IB device on the Module. A Managed Chassis may contain a switch which handles the send/receive of the Baseboard Management MADs. The MADs are addressed using the LID of the switch.

16.2.1 MAD FORMAT

C16-10: The datagrams in the Baseboard management class shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 175 Baseboard Management MAD Format on page 783 and Table 201 Baseboard Management MAD Fields on page 784 below.

Figure 175 Baseboard Management MAD Format

bytes	
0	Common MAD Header
...	
20	
24	B_Key
28	
32	Reserved
...	
60	

Figure 175 Baseboard Management MAD Format

bytes	
64	Data
...	
252	

Table 201 Baseboard Management MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD as described in 13.4.2 Management Datagram Format on page 604
B_Key	8 bytes	BM specific key. See 16.2.4 B_Key General Use on page 790 for definition and use.
Reserved	32 bytes	This field is reserved and shall be set to zeroes.
Baseboard Management Data	192 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.2.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. No class-specific bits are defined.

Table 202 Baseboard Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617
8-15	-	Class-specific bits are reserved

16.2.2 METHODS

The Baseboard Management class uses a subset of the common methods described in 13.4.5 Management Class Methods on page 607.

Table 203 Baseboard Management Methods

Method Type	Value	Description
BMGet()	0x01	Request a get (read) of an attribute.
BMSet()	0x02	Request a set (write) of an attribute.
BMGetResp()	0x81	Response from a get or set request.
BMSend()	0x03	Send Baseboard Management attribute (this can be the attribute for an encapsulated Baseboard Management request [command] or response.).

Table 203 Baseboard Management Methods

Method Type	Value	Description
BMTrap()	0x05	Notify an event occurred.
BMTrapRepress()	0x07	Block repetition of notification.

Here are some examples of transactions. Semantics are described in Volume 2A, Chapter 9.

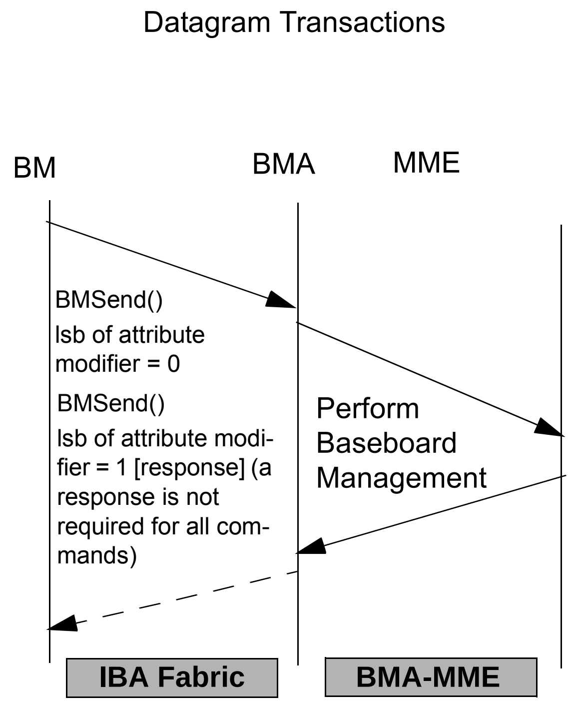


Figure 176 BM initiated IB-ML cmd

Requests and response capabilities are symmetric. I.e. BMSend() may be used to send a request from the MME as well as a response. Similarly, the Baseboard Manager may use BMSend to deliver a response. Figure 177 IB-ML initiated cmd on page 786, illustrates the path that would be taken if a CME generated a request to the Baseboard Manager by delivering the request via a module's IB-ML -to- IB functionality.

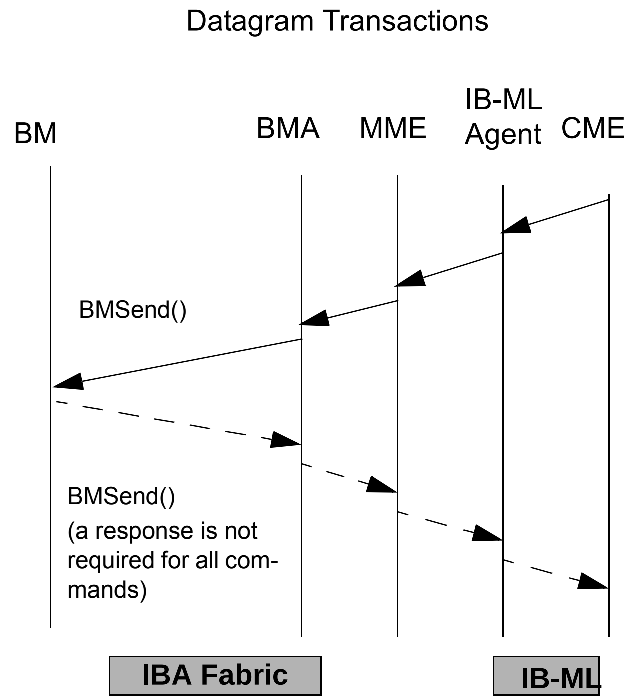


Figure 177 IB-ML initiated cmd

16.2.3 ATTRIBUTES

Table 204 Baseboard Management Attributes on page 786 summarizes the attributes used by the Baseboard Management class.

Table 204 Baseboard Management Attributes

Attribute Name	Attribute ID	Attribute Modifier[a]	Description
ClassPortInfo	0x0001	0x00000000	General and port-specific information for the BM class.
Notice	0x0002	0x00000000	Information regarding a Trap.
BKeyInfo	0x0010	0x00000000	B_Key information for the node.
WriteVPD	0x0020	0x00000000 / 0x00000001	See Hardware Management chapter in Volume 2 for this and following attributes.
ReadVPD	0x0021	0x00000000 / 0x00000001	
ResetIBML	0x0022	0x00000000 / 0x00000001	
SetModulePMControl	0x0023	0x00000000 / 0x00000001	

Table 204 Baseboard Management Attributes

Attribute Name	Attribute ID	Attribute Modifier[a]	Description
GetModulePMControl	0x0024	0x00000000 / 0x00000001	
SetUnitPMControl	0x0025	0x00000000 / 0x00000001	
GetUnitPMControl	0x0026	0x00000000 / 0x00000001	
SetIOCPMControl	0x0027	0x00000000 / 0x00000001	
GetIOCPMControl	0x0028	0x00000000 / 0x00000001	
SetModuleState	0x0029	0x00000000 / 0x00000001	
SetModuleAttention	0x002A	0x00000000 / 0x00000001	
GetModuleStatus	0x002B	0x00000000 / 0x00000001	
IB2IBML	0x002C	0x00000000 / 0x00000001	
IB2CME	0x002D	0x00000000 / 0x00000001	
IB2MME	0x002E	0x00000000 / 0x00000001	
OEM	0x002F	0x00000000 / 0x00000001	

a. Where two attribute modifiers are listed, the least significant bit of the attribute Modifier is used to identify BMSend() requests (0) from responses. Refer to InfiniBand Architecture Specification, Volume 2, Chapter “Hardware Management” for more details.

Table 205 Baseboard Management Attribute / Method Map

Attribute Name	BMGet	BMSet	BMSend	BMTrap
ClassPortInfo	x	x		
Notice	x	x		x
BKeyInfo	x	x		
WriteVPD			x	
ReadVPD			x	
ResetIBML			x	
SetModulePMControl			x	
GetModulePMControl			x	
SetUnitPMControl			x	
GetUnitPMControl			x	
SetIOCPMControl			x	
GetIOCPMControl			x	
SetModuleState			x	

Table 205 Baseboard Management Attribute / Method Map

Attribute Name	BMGet	BMSet	BMSend	BMTrap
SetModuleAttention			x	
GetModuleStatus			x	
IB2IBML			x	
IB2CME			x	
IB2MME			x	
OEM			x	

16.2.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. In addition, bit 8 of the CapabilityMask component is defined:

Table 206 Baseboard Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619
8	IsIBMLSupported	Direct Access to IB-ML is supported
9	IsBKeyNVRAM	B_Key is in NVRAM
10-15		Reserved

16.2.3.2 NOTICE

The Notice attribute is described in 13.4.8.2 Notice on page 622. It is used for one optional generic trap.

Table 207 Baseboard Management Traps

Name	Type	Number	DataDetails
BKeyViolation	Security	259	*Bad B_Key,* <B_Key> *from* <LIDADDR>/<GIDADDR>/<QP> *attempted* <METHOD> with <ATTRIBUTEID> and <ATTRIBUTEMODIFIER>.
BMTraps	Various	260	Defined in InfiniBand Architecture Specification, Volume 2, Chapter “Hardware Management”. Several kind of traps are encoded by this number, differentiated by the content of DataDetails

o16-3: The BKeyViolation trap uses the following layout for the DataDetails component of the Notice attribute, see Table 208 Notice DataDetails

For Trap 259 on page 789. Fields shall be filled with the information corresponding to the description of a given trap.

Table 208 Notice DataDetails For Trap 259

Field	Length(bits)	Description
LIDADDR	16	Local Identifier
METHOD	8	Method
Reserved1	8	Shall be filled with zeroes
ATTRIBUTEID	16	Attribute ID
ATTRIBUTE MODIFIER	32	Attribute Modifier
Reserved2	8	Shall be filled with zeroes
QP	24	Queue Pair
BKEY	64	B_Key
GIDADDR	128	Global Identifier. If no GRH is present in the offending packet, this field shall be filled with zeroes.
Padding	128	Shall be ignored on read. Content is unspecified.

16.2.3.3 BKEYINFO

Table 209 BKeyInfo

Component	Access	Length(bits)	Description
B_Key	RW	64	The 8-byte Baseboard Management key used in all BM MADs by all valid BMs. A value of 0 means no B_Key check is ever done by the BMA.
B_KeyProtectBit	RW	1	See 16.2.4.3 B_Key Operation on page 791 for details.
Reserved	RW	15	Shall be set to zeroes.
B_KeyLeasePeriod	RW	16	Timer value used to indicate how long the B_Key Protection bit is to remain non zero after a BMSet(BKeyInfo) MAD that failed a B_Key check is dropped. The value of the timer indicates the number of seconds for the lease period. With a 16 bit counter, the period can range from one second to approximately 18 hours. 0 shall mean infinite. See 16.2.4.5 B_Key Recovery on page 792 for details.
B_KeyViolations	RO	16	Number of MADs that have been received at this node since power-on or reset that have been dropped due to a failed B_Key check if such a counter is implemented. Otherwise this shall be 0xFFFF.

16.2.3.4 IB-ML ATTRIBUTES

See *InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management", section "Management Commands"* for a description of the BM class specific attributes and their format.

16.2.4 B_KEY GENERAL USE

The BM includes the Baseboard Management Key (B_Key) in the BM MAD to obtain authorization. The B_Key is used to authenticate a trusted source. This model assumes that the fabric has some level of physical security.

16.2.4.1 B_KEY ASSUMPTIONS

1) To use the correct key for each node, the BM or a higher-level B_Key manager keeps track of the keys for the nodes that it is managing.

2) If a backup BM exists, it shares the B_Keys for ease of fail-over.

3) A BM may have exclusive access to a set of nodes, by using a B_Key which is only known by that BM and those particular nodes. Since nodes reply to Baseboard Manager's requests using their own B_Key, if a Baseboard Manager assigns more than one B_Key to devices on its management domain, it needs to run on an HCA whose BMA can respond to all such B_Keys.

4) The BM sets the B_Key, the B_Key Protection bit, and the B_Key lease period in the BKeyInfo Attribute with one BMSet(BkeyInfo)

MAD. A successful completion of this assignment indicates to the BM that it has taken ownership of the node.

16.2.4.2 B_KEY PROTECTION SCOPE

Each BMA (Baseboard Management Agent) in a node has one B_Key. Table 210 on page 791 shows the scope protected by that B_Key. The semantics are explicitly defined in *InfiniBand Architecture Specification, Volume 2, Chapter "Hardware Management"*.

Table 210 B_Key Protection Scope

Source	Targeted Entity	Protection
BM	Read and Writes to - ClassPortInfo (e.g. BM LID in TrapLID) - BKeyInfo (e.g. B_Key, B_Key Protection Bit)	yes
BM	Attributes causing reads from and writes to IB-ML - ModuleInfo[a] - IB-module Specific Data - ChassisInfo[b] - CME[b] - Other IB-ML devices	yes
IB-ML Managed Unit	Attributes causing reads from and writes to IB-ML - IB-module VPD - IB-module Specific Data - ChassisInfo[b] - Other IB-ML devices	no

a. The IB-Module vendor protects the factory-programmed portion of ModuleInfo against writes even if a proper B_Key is provided.
b. The Chassis vendor protects the factory-programmed portion of ChassisInfo against writes even if a proper B_Key is provided. If further protection is desired, the CME or the Chassis provides it.

16.2.4.3 B_KEY OPERATION

C16-11: The BMA shall check the B_Key contained in incoming MADs.

The success and effect of the check depends on the value of the B_Key and B_Key Protection bit of the BMA and on the method and attribute contained in the incoming MAD.

Table 211 B_Key Check

BMA's B_Key	BMA's B_Key Protection Bit	MAD's method	Success
zero	any	any	yes
non-zero	any	BMSet(), BMSend()	if MAD's B_Key equals BMA's B_Key

Table 211 B_Key Check

BMA's B_Key	BMA's B_Key Protection Bit	MAD's method	Success
non-zero	0	BMGet()	yes
non-zero	1	BMGet()	yes[a]

a. Even though the check succeeds, the B_Key value in the BKeyInfo attribute shall be returned as zero.

C16-12: If B_Key check fails, the BMA shall:

1) Drop the MAD.
2) Increment a B_Key Violation counter if supported.
3) Send a BKeyViolation trap if traps are supported by the BMA.
4) Start a countdown timer with the B_Key lease period value.

16.2.4.4 B_KEY INITIALIZATION

C16-13: At power up or reset, the B_Key, B_Key Protection bit and B_Key lease period shall be set to zero if NVRAM is not used; otherwise, they shall be set to the values stored in NVRAM.

Using a BMSet(BKeyInfo), the BM may assign the subsequent B_Key, B_Key Protection bit and B_Key lease period.

16.2.4.5 B_KEY RECOVERY

The B_Key lease period timer starts when a B_Key check fails. At this time, the node sends a trap to the BM (if traps are supported and if the BM stored its information in the trap components of the ClassPortInfo attribute). This trap serves as a request to the BM to refresh the lease period by issuing a BMSet(BKeyInfo). A successful BMSet(BKeyInfo) will stop the timer and will rearm it.

If the BM that originally set the B_Key has gone away, then the lease period expires-- clearing the B_Key Protection bit and allowing anyone to read (and then set) the B_Key.

In the case where a node starts with NVRAM B_Key and B_Key Protection bits set and the TrapLID is zero (because no BM has come around to set it), the node has no BM to send the trap to. In this case, the node does not send the trap and the lease period timer will expire, causing eventual take over by a new BM.

With the BMGet(BKeyInfo), any BM can detect whether a B_Key is set (although hidden) based on the B_Key Protection bit. If the B_Key Protection bit is set, the B_Key is set and hidden. Otherwise the returned B_Key is the real one even if it is zero.

16.2.4.6 LEVELS OF PROTECTION

There are four different protection levels based on the B_Key, depending on the system requirements.

Table 212 Protection Levels

B_Key	B_Key Protection bit	B_Key Lease Period	Description
0	any	any	No protection provided. Any BM can issue sets and sends.
non-zero	0	n/a	Protection provided, but allows BMs to read the B_Key in the node.
non-zero	1	non-zero	Protection provided and does not allow anyone to read the B_Key in the node until the lease period has expired. The B_Key lease period is a mechanism to allow the B_Key to be protected only for a given amount of time.
non-zero	1	0	Protection provided and does not allow the B_Key in the node to be read by other BMs. It must be noted that if the lease period was set to 0 (infinite) and the BM that set it dies, there is no possibilities for other BMs to ever read it. So if the B_Key is not provided by some unspecified way to the other BMs, the BMA of this node will never be accessible again.

16.3 DEVICE MANAGEMENT

The Device Management Agent is optional.

IO Devices and I/O controllers (IOC) are not directly connected to the IBA fabric. An I/O Unit (IOU) containing one or more IOCs is attached to the fabric via a TCA. The TCA is responsible for receiving packets from the fabric and delivering complete, valid messages to IOCs, and vice-versa. The TCA might use memory resources supplied by the IOC to assemble the packet and notifies the IOC when the complete packet is available for consumption. IOC is then responsible for executing I/O requests such as network sends and receives or disk reads and writes over a device specific interface such as Ethernet or SCSI.

This chapter does not address direct management of end devices such as disk drives but focuses on the infrastructure, related methods, data formats and attributes to support IOU/IOC management over the fabric. This chapter defines mechanisms to send and receive device management packets between two fabric attachment points such as a HCA and a TCA. The mechanisms required to translate MADs into a format that the end devices understand and how the data is delivered and retrieved from an end device is device-specific and therefore is not addressed in this specification.

The IBA is based on message passing. For IOU and IOC, the messages fall into three classes: fabric configuration, unit management/configuration, and I/O transaction:

- Fabric configuration messages that are processed by the Subnet Management Agent (SMA) are defined in Chapter 14.
- Messages specific to configuring and managing a device that are received through the General Services Interface (GSI) are described in this section.
- IO transaction messages are not defined in this document. I/O transaction messages include those messages used by an initiator to request I/O services from an IOC, messages containing user or application data, and messages used by the IOC to provide a completion notification (ending status) to the requestor. Also included in this class are in-band configuration messages (parameters, etc.) directed only to an IOC, and not to the larger IOU as a whole. These messages travel as I/O requests but perform management functions specific to the I/O controller.

Although this chapter tends to use language implying that an IOU "contains" IOCs, there are no restrictions on how IOCs are connected to, or served by, the TCA. Figure 178 Architectural Model for an I/O Unit on page 794 provides the architectural and connection models for an IOU, consisting of a TCA and one or more IOCs.

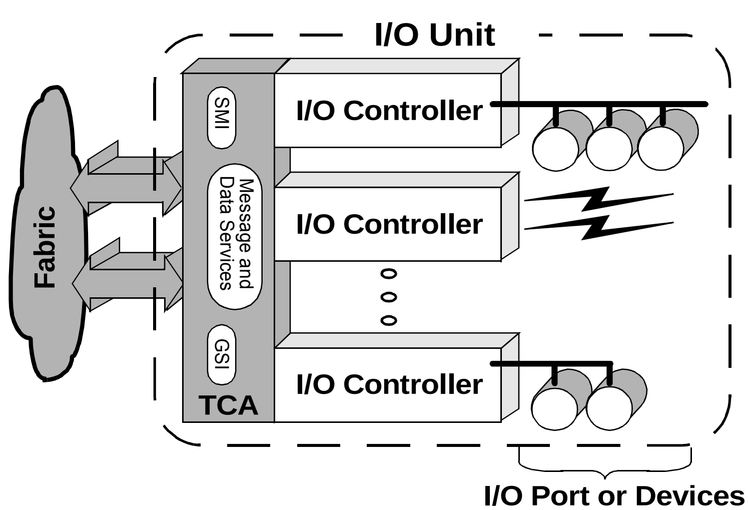


Figure 178 Architectural Model for an I/O Unit

16.3.1 MAD FORMAT

o16-4: The datagrams in the Device Management class shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 179 Device Management MAD Format on page 795 and Table 213 Device Management MAD Fields on page 795 below.

Figure 179 Device Management MAD Format

bytes	
0	Common MAD Header
...	
20	
24	Reserved
...	
60	
64	Data
...	
252	

Table 213 Device Management MAD Fields

Field	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 604
Reserved	40 bytes	Shall be set to zeroes.
DevMgt Data	192 bytes	192 bytes of Device Management payload. The structure and content depends upon the Method, Attribute and Attribute Modifier fields in the header.

16.3.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. Some class-specific bits are defined.

Table 214 Device Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617
8	NoResponse	IOC Not responding
9	NoServiceEntries	Service Entries are not supported

Table 214 Device Management Status Field

Bits	Name	Meaning
10-14	-	Reserved
15	GeneralFailure	IOC General Failure

16.3.2 METHODS

Among the services that a TCA provides to an initiating client is a mechanism to deliver detailed information about the I/O resources (e.g. IOCs) supported by the IOU. This information transcends a simple count of the number of IOCs supported to provide details of each IOC such as a GUID, a vendor-unique ID, product revision levels, and other information that is specific to a given IOC. The purpose of the detailed information is to let a system configuration manager allocate the IOU's resources to various clients located on the IBA fabric, and to provide a common way for host resource managers to determine the characteristics of IOUs and IOCs. This allows the proper driver to be associated with each controller.

The profiles are requested and returned through the GSI, which is an unreliable datagram service. The actual access QP and DLID may be redirected by the GSI. The IOUnitInfo attribute contains information on the number of IOCs the unit can support (IOUnitInfo:MaxControllers). This value is the length of the IOUnitInfo:ControllerList, which has an entry for every possible controller "slot" (which may be physical or logical). Each entry in the ControllerList component shows whether a controller is present. For each controller, the IOControllerProfile attribute contains information such as the type of controller and the number of connections the IOC can support. Each controller has a ServiceEntries attribute associated with it. ServiceEntries is a table of ServiceIDs that the controller advertises to its clients. The format of the IOUnitInfo, IOControllerProfile and ServiceEntries structures are defined in 16.3.3 Attributes on page 797.

Table 215 Device Management Methods

Method Type	Value	Description
DevMgtGet()	0x01	Request an IOU to return (read) Device Management class attributes such as profile or a list of controllers currently installed.
DevMgtSet()	0x02	Request an IOU to set (write) an attribute. The object will issue a DevMgtGetResp() as a response.
DevMgtGetResp()	0x81	IOU responds to an attribute Get or Set request.
DevMgtTrap()	0x05	Unsolicited datagram sent to the Device Management entity. Contains the Notice Attribute as defined in 13.4.8.2 Notice on page 622 to identify the trap.
DevMgtTrapRepress	0x07	Block repetition of notification.

16.3.3 ATTRIBUTES

This section specifies the format of the attributes used for managing the IOU. Messages used as part of I/O transactions are not specified in this document. The term “device” refers to actual devices sitting behind IOCs. The way they are numbered is implementation-specific.

Table 216 Device Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x0000_0000	See 13.4.8.1 ClassPortInfo on page 619
Notice	0x0002	0x0000_0000-0xFFFF_FFFF	See 13.4.8.2 Notice on page 622
IOUnitInfo	0x0010	0x0000_0000	List of all IOCs present in a given IOU. Each IOU may support up to 0xFF controllers.
IOControllerProfile	0x0011	0x0000_0001-0x0000_00FF	IOC Profile Information. Attribute Modifier identifies the IOC.
ServiceEntries	0x0012	0x0001_0000-0x00FF_FFFF	List of supported services and their associated Service IDs. Each IOC has a table with at most 0x100 ServiceEntries. The attribute modifier is structured as follows: - the upper 16 bits identify the IOC - the lower 16 bits specify a range of up to four Service Entries to be retrieved. The first 8 bits of the lower 16 bits specify the beginning and the last 8 bits the end of the range.
Reserved	0x0013-0x001F	0x0000_0000-0xFFFF_FFFF	Reserved
DiagnosticTimeout	0x0020	0x0000_0001 - 0xFFFF_FFFF	Response indicates maximum time for completion of diagnostic test. Target device is identified by the Attribute Modifier. Tests not completing within this period may indicate device failure. Specified in multiples of milliseconds.
PrepareToTest	0x0021	0x0000_0001 - 0xFFFF_FFFF	A Set with this Attribute instructs the device specified by the Attribute Modifier to prepare for diagnostic test. A Get of this Attribute will result in the appropriate Response Status being set as follows: 0x0000 = Ready for diagnostic test 0x0100 = Invalid Attribute Modifier 0x0200 = Device not ready 0x0400 = Device not responding 0x0800 = Diagnostics not supported 0x1000 - 0x8000 = Reserved
TestDeviceOnce	0x0022	0x0000_0001 - 0xFFFF_FFFF	A Set instructs the device specified by the Attribute Modifier to initiate a single diagnostic test and run it once. Vendor-unique attribute values may be defined to initiate specific test instructions.

Table 216 Device Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
TestDeviceLoop	0x0023	0x0000_0001 - 0xFFFF_FFFF	A Set instructs the device specified by the Attribute Modifier to initiate a single diagnostic test and run it continuously in a loop. Vendor-unique attribute values may be defined to initiate specific test instructions.
DiagCode	0x0024	0x0000_0001 - 0xFFFF_FFFF	Vendor-specific Diagnostic information for the device specified by the Attribute Modifier. See 14.2.5.6.1 Interpretation of DiagCode on page 671.
Reserved	0x0025-0xFEFF	0x0000_0000 - 0xFFFF_FFFF	Reserved
Vendor specific	0xFF00-0xFFFF	0x0000_0000 - 0xFFFF_FFFF	Vendor-unique attribute values may be defined to initiate specific test instructions.

Table 217 Device Management Attribute / Method Map

Attribute Name	DevMgtGet	DevMgtSet	DevMgtTrap
ClassPortInfo	x	x	
Notice	x	x	x
IOUnitInfo	x		
IOControllerProfile	x		
ServiceEntries	x		
DiagnosticTimeout	x		
PrepareToTest	x	x	
TestDeviceOnce		x	
TestDeviceLoop		x	
DiagCode	x		

16.3.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. No class-specific bits are defined.

Table 218 Device Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619

Table 218 Device Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
8-15	-	Class-specific bits are reserved

16.3.3.2 NOTICE

The Notice attribute is described in 13.4.8.2 Notice on page 622. It is used for one optional generic trap.

Table 219 Device Management Traps

Name	Type	Number	DataDetails
ReadyToTest	Informational	514	*Device* <DEVICE> *readiness is* <STATUS>*, where status is the same as would have been returned by a Get(PrepareToTest) with device as the attribute modifier.*

o16-5: Device Management Traps use the following layout for the DataDetails component of the Notice attribute, see Table 220 Notice DataDetails For Trap 514 on page 799. Fields shall be filled with the information corresponding to the description of a given trap.

Table 220 Notice DataDetails For Trap 514

Field	Length(bits)	Description
STATUS	16	Readiness status
DEVICE	32	Device number
Padding	384	Shall be ignored on read. Content is unspecified.

16.3.3.3 IOUNITINFO

Table 221 IOUnitInfo

Component	Access	Length(bits)	Description
Change_ID	RO	16	Incremented, with rollover, by any change to ControllerList.
Max Controllers	RO	8	Number of slots in ControllerList.
Reserved	RO	7	Reserved for future use.
Option ROM	RO	1	Indicates presence of Option ROM. 1 = Present; 0 = Absent.

Table 221 IOUnitInfo

Component	Access	Length(bits)	Description
ControllerList	RO	1024	A series of 4-bit nibbles with each representing a slot in the IOU. Each 4-bit nibble can take the following values: - 0x0 = IOC not installed - 0x1 = IOC present - 0x2-0xe = reserved - 0xf = slot does not exist Bits 7-4 of the first byte (lowest offset) represent slot 1, bits 3-0 represent slot 2, bits 7-4 of the second byte represent slot 3, bits 3-0 represent slot 4, and so on.

16.3.3.4 IOCONTROLLERPROFILE

Table 222 IOControllerProfile

Component	Access	Length(bits)	Description
GUID	RO	64	An EUI-64 GUID used to uniquely identify the controller. This could be the same one as the Node/Port GUID if there is only one controller.
VendorID	RO	24	IO controller vendor ID, IEEE format
Reserved	RO	8	Reserved for proper alignment.
DeviceID	RO	32	A number assigned by the vendor to identify the type of controller. This can be used by an Operating System to select a device driver.
Device Version	RO	16	A number assigned by the vendor to identify the device version.
Reserved	RO	16	Reserved for proper alignment.
Subsystem VendorID	RO	24	ID of the vendor of the enclosure, if any, in which the I/O controller resides in IEEE format; otherwise zero.
Reserved	RO	8	Reserved for proper alignment.
SubsystemID	RO	32	A number identifying the subsystem where the controller resides.
IO Class	RO	16	0x0000-0xfffe = Reserved pending I/O class specification approval. 0xffff = Vendor-specific
IO Subclass	RO	16	0x0000-0xfffe = Reserved pending I/O subclass specification approval. 0xffff = Vendor-specific This must be set to 0xffff if the I/O Class component is set to 0xffff.

Table 222 IOControllerProfile

Component	Access	Length(bits)	Description
Protocol	RO	16	0x0000-0xfffe = Reserved pending I/O protocol specification approval. 0xffff = Vendor-specific This must be set to 0xffff if the I/O Class component is set to 0xffff.
Protocol Version	RO	16	Protocol specific.
Service Connections	RO	16	Number of service connections controller can support.
Initiators Supported	RO	16	Number of initiators that this IOC can support.
Send Message Depth	RO	16	Maximum Depth of the Send Message Queue.
RDMA Read Depth	RO	16	Maximum Depth of the per-channel RDMA Read Queue.
Send Message Size	RO	32	Maximum size of Send Messages in bytes.
RDMA Transfer Size	RO	32	Maximum size of outbound RDMA transfers initiated by the IOC - in bytes.
Controller Operations Capability Mask	RO	8	Supported operation types of this I/O controller. A bit set to 1 for affirmation of supported capability. Bit: Name; Description 0: ST; Send Messages To IOCs 1: SF; Send Messages From IOCs 2: RT; RDMA Read Requests To IOCs 3: RF; RDMA Read Requests From IOCs 4: WT; RDMA Write Requests To IOCs 5: WF; RDMA Write Requests From IOCs 6: AT; Atomic Operations To IOCs 7: AF; Atomic Operations From IOCs
Controller Services Capability Mask	RO	8	Supported operation types of this I/O controller. A bit set to 1 for affirmation of supported capability. Bit: Name; Description 0: CS; Console Services 1: SBWP; Storage Boot Wire Protocol 2: NBWP; Network Boot Wire Protocol 3-7: Reserved, For future services
Service Entries	RO	8	Number of entries in the ServiceEntries table.
Reserved	RO	72	Reserved for future use.
ID String	RO	512	UTF-8 encoded string for identifying the controller to operator.

16.3.3.5 SERVICEENTRIES

Table 223 ServiceEntries

Component	Access	Length(bits)	Description
ServiceName_1	RO	320	String of Service name in UTF-8 format.
ServiceID_1	RO	64	An identifier of the associated Service.
ServiceName_2	RO	320	String of Service name in UTF-8 format.
ServiceID_2	RO	64	An identifier of the associated Service.
ServiceName_3	RO	320	String of Service name in UTF-8 format.
ServiceID_3	RO	64	An identifier of the associated Service.
ServiceName_4	RO	320	String of Service name in UTF-8 format.
ServiceID_4	RO	64	An identifier of the associated Service.

16.3.3.6 DIAGNOSTICTIMEOUT

Table 224 DiagnosticTimeout

Component	Access	Length(bits)	Description
MaxDiagTime	RO	32	Maximum time to finish a diagnostic operation in milli-seconds

16.3.3.7 PREPARETOTEST

Table 225 PrepareToTest

Component	Access	Length(bits)	Description
-	-	-	This attribute does not have any components

16.3.3.8 TESTDEVICEONCE

Table 226 TestDeviceOnce

Component	Access	Length(bits)	Description
-	-	-	This attribute does not have any components

16.3.3.9 TESTDEVICELOOP

Table 227 TestDeviceLoop

Component	Access	Length(bits)	Description
-	-	-	This attribute does not have any components

16.3.3.10 DIAGCODE

Table 228 DiagCode

Component	Access	Length(bits)	Description
Vendor-specific			

16.3.4 DEVICE DIAGNOSTIC FRAMEWORK

Device Diagnostics allows the identification of faults in devices behind the target channel adapter. As such, it complements other sections of this specification that describe how problems at the fabric and node level may be identified and isolated.

The device diagnostic framework is intended to support tests within an active fabric. It is versatile enough, however, to accommodate vendor-unique approaches that may include retrieval of power-on data. It should be noted that some, and perhaps most, devices may not permit simultaneous use of I/O transaction messages and diagnostics. Unless data is flushed from internal buffers, for example, corruption or loss of user data might occur. Further, it is expected that the diagnostics tests would require setting the device to an initial, known state. For that reason, provision will be made to put the device into a “ready” state prior to test, which will likely cause I/O transactions to be held off. This may, in turn, cause established connections to time out, and other management notices to be sent.

In general, device diagnostics should be used with great care, and with full understanding of the potential impact to I/O transactions to the target device. It is best used during periods of initial configuration, major maintenance, or as a tool of last resort.

16.3.4.1 BEHAVIORS

The Device Management class of MADs (see 16.3 Device Management on page 793) is used for diagnostics. Within that class, standard methods as defined in Table 215 Device Management Methods on page 796 are utilized. Attributes specific to device diagnostics are defined by which vendor-supplied tests may be invoked, and the results of completed tests then determined. Attribute Modifiers are used to indicate the Port Number of the device under test. Results are reported in a format consistent with the 16-bit DiagCode used for the Node Diagnostics (see 14.2.5.6.1 Interpretation of DiagCode on page 671).

The PrepareToTest attribute within the DevMgtSet method places the device into a test-ready state. The time required to complete this step is not predictable, as it may involve flushing data from cache memory, reinitializing SCSI ports, etc. The device indicates its readiness for test by signaling the IOU to send an Informational Trap.

Alternatively, a Get method on this attribute will return information pertaining to the specific device's ability or readiness for test. This allows the status to be polled on a periodic basis, or to determine that the device does not support diagnostic tests.

Two modes are provided for initiating diagnostics: single test mode and continuous test mode. In single test mode, a single test sequence is initiated by setting a non-zero value for the TestDevice attribute, with MSB=0. Once initiated, this vendor-defined test will run to completion. Because tests will vary by device technology and by vendor, the time-to-completion is inherently unpredictable. To detect errant devices which are unable to complete their diagnostic test, a DiagnosticTimeout attribute may be retrieved in advance of test initiation, which indicates the maximum allowable period for completion. Results of the completed diagnostic test are obtained through the DiagCode attribute of the GetResp method.

The continuous test mode can assist in detecting problems that are transient in nature, be used to initiate endurance-related tests. The continuous-test mode is initiated by setting a non-zero value for the TestDevice attribute, with MSB=1. Results of the last completed diagnostic test are obtained through the DiagCode attribute of the GetResp method.

Interpretation of results obtained through the GetResp method is vendor-specific.

It is beyond the scope of this specification to define a set of white-box, technology-specific diagnostic tests. Rather, the intent is to allow initiation of a vendor-supplied test sequence, for which the expected outcome would be either success or failure. The DiagCode format, however, allows flexibility for the vendor to provide specific, coded information about the test results.

16.4 SNMP TUNNELING

The SNMP Tunneling Agent is optional.

This section describes the Management Datagrams used to report native SNMP tunneling over the IBA.

SNMP, or Simple Network Management Protocol, consists of a set of standards for network management, a protocol, and a database specification to uniformly address managed information objects structured in a format called MIB-II (Management Information Base version 2). SNMP was originally specified in RFC1157, and later, RFC1902 for SNMP v2.

The structure of management information was originally laid out in RFC1155; the current MIB-II standard resides in RFC 1213. The supported RFCs to which this document references will be RFC1902-1908, for SNMPv2, although SNMPv2 supports SNMPv1, and RFC1213, for the MIB-II standard.

SNMP tunneling is a supported option to the InfiniBand architecture as a Management Datagram service. Devices advertise support for the SNMP tunneling service by use of the IsSNMPSupported Capability in PortInfo Attribute. If the value is non-zero a given device may be queried via the GSI for the QP and LID to access the SNMP service. Note that this capability allows for another port to supply SNMP tunneling services by proxy.

This section describes the required class-dependent behavior of the datagrams in this class.

16.4.1 MAD FORMAT

o16-6: The datagrams in the SNMP tunneling class shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 180 SNMP Tunneling MAD Format on page 805 and Table 229 SNMP Tunneling MAD Fields on page 806 below.

Figure 180 SNMP Tunneling MAD Format

bytes			
0	Common MAD Header		
...			
20			
24	Reserved		
...			
52			
56	Raddress		
60	Payload Length	Segment Number	Source LID

Figure 180 SNMP Tunneling MAD Format

bytes	
64	Data
...	
255	

Table 229 SNMP Tunneling MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 604
Reserved	32 bytes	Set to all 0.
Raddress	4 bytes	Opaque address field that is used by SNMP agent to forward SNMP packets using SNMP redirect features.
Payload Length	1 byte	Number of valid data bytes in entire SNMP packet being transferred.
Segment Number	1 byte	Segment number of a segmented SNMP packet.
Source LID	2 bytes	Local address of the SNMP packet sender.
Data	192 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.4.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. No class-specific bits are defined.

Table 230 SNMP Tunneling Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617
8-15	-	Class-specific bits are reserved

16.4.2 METHODS

This class utilizes the common methods described in 13.4.5 Management Class Methods on page 607.

Table 231 SNMP Tunneling Methods

Method Type	Value	Description
SnmpGet()	0x01	Request a get (read) of an Attribute.

Table 231 SNMP Tunneling Methods

Method Type	Value	Description
SnmpSet()	0x02	Request a set (write) of an Attribute.
SnmpGetResp()	0x81	Response from a get or set request.
SnmpSend()	0x03	Send an Attribute to a node.

16.4.3 ATTRIBUTES

Table 232 SNMP Tunneling Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description	Length
ClassPortInfo	0x0001	0x0000_0000	See 13.4.8.1 ClassPortInfo on page 619	
CommunityInfo	0x0010	0x0000_0000 0x0000_0001 0x0000_0002 0x0000_0003	Community Name Data Store	64 bytes
PduInfo	0x0011	0x0000_0001	First SNMP segment	192 bytes
		0x0000_0002	Intermediate SNMP segment	
		0x0000_0003	Last SNMP segment	
		0x0000_0004	First and Last SNMP segment	
		0x8000_0001	First redirected SNMP segment	
		0x8000_0002	Intermediate redirected SNMP segment	
		0x8000_0003	Last redirected SNMP segment	
		0x8000_0004	First and Last redirected SNMP segment	

Table 233 SNMP Tunneling Attribute / Method Map

Attribute Name	SnmpGet	SnmpSet	SnmpSend
ClassPortInfo	x	x	
CommunityInfo			x
PDUInfo			x

16.4.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. No class-specific bits are defined.

Table 234 SNMP Tunneling ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619
8-15	-	Class-specific bits are reserved

16.4.3.2 COMMUNITYINFO

Table 235 CommunityInfo

Component	Settability	Length(bits)	Description
Community Name	RW	512	Community Name, used for authentication. The SNMP standard specifies a 255 byte community name field in the SNMP packet. This field is used for authentication by the SNMP protocol. This Attribute stores the community name in four 64 byte segments. The SNMP agent residing in the Endnode concatenates the four segments in order 0-3 to form the 255 byte string. UTF-8 encoding shall be used.

16.4.3.3 PDUINFO

Table 236 PduInfo

Component	Settability	Length(bits)	Description
PduData	RW	1536	SNMP data segment

16.4.4 OPERATIONS

Figure 181 depicts the SNMP PDU format (shown in network byte order, as published in IETF publications). This is abstracted on top of the MAD datagram.

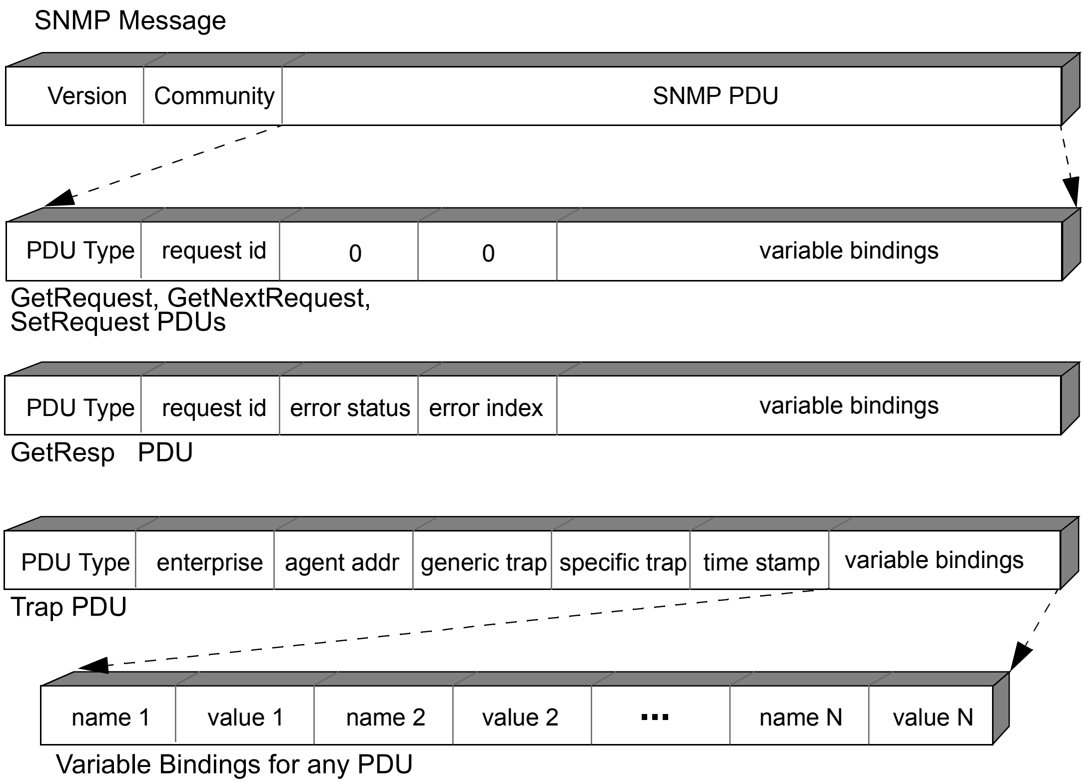


Figure 181 SNMP PDU Format

The maximum object definable in a MIB is 256 bytes, but the maximum payload available in a SNMP datagram is 192 bytes. Because SNMP cannot be redefined to suit the MAD datagram, the InfiniBand SNMP transport provides segmentation/reassembly.

A packet consists of one or more segments. If necessary, a packet will be segmented at the source, transmitted, and reassembled at the target. Using the SNMP datagram, the source specifies the SnmpSend method, PduInfo Attribute, then sets the Attribute modifier, segment number (if the packet is segmented), and the payload length fields to delimit and account for segments.

o16-7: When SNMP packets must be segmented into multiple MADs, the data field of all but the last MAD transferred shall be completely filled (192 bytes of data).

o16-8: If any segment of a multiMAD transfer is not received within the timeout as specified in 13.4.6.3 Timeout/Timer Usage on page 615, then that entire MAD shall be discarded.

o16-9: The Transaction ID field of all the MADs of the SNMP packet shall be the same, and shall conform to the uniqueness of Transaction IDs as described in 13.4.6.4 TransactionID usage on page 616.

Because the destination is already known by the sender of the MAD packet, there is no need to include it in the MAD packet. However, because the sender may be expecting a response from the agent receiving this SNMP request, the original source LID is provided so the agent knows where to send a reply.

The maximum number of bytes transmittable in a single UDP Packet is slightly over 8192. The SNMP management class can transfer a packet of up to 49,152 bytes. This is enough to accommodate any incoming SNMP/UDP packet and allow for flexibility if management packets arrive from a transport other than TCP/UDP.

o16-10: When the pieces are reassembled, the SNMP Message shall be extracted and passed up to the agent or manager for processing.

If a MAD is marked *First and Last* with the attribute modifier, it is the only segment in the packet. No segmentation has occurred so no reassembly is required, extract and pass it up.

If SNMP Redirect is specified in the attribute modifier, the packet is meant for a target managed by the proxy agent processing the packet. The proxy agent will need to parse the packet to extract the *Raddress* value of the final destination to reformat the PDU for further transport along the new interconnect.

16.4.4.1 SNMP TARGETS FOR BEYOND THE INFINIBAND ENDNODE

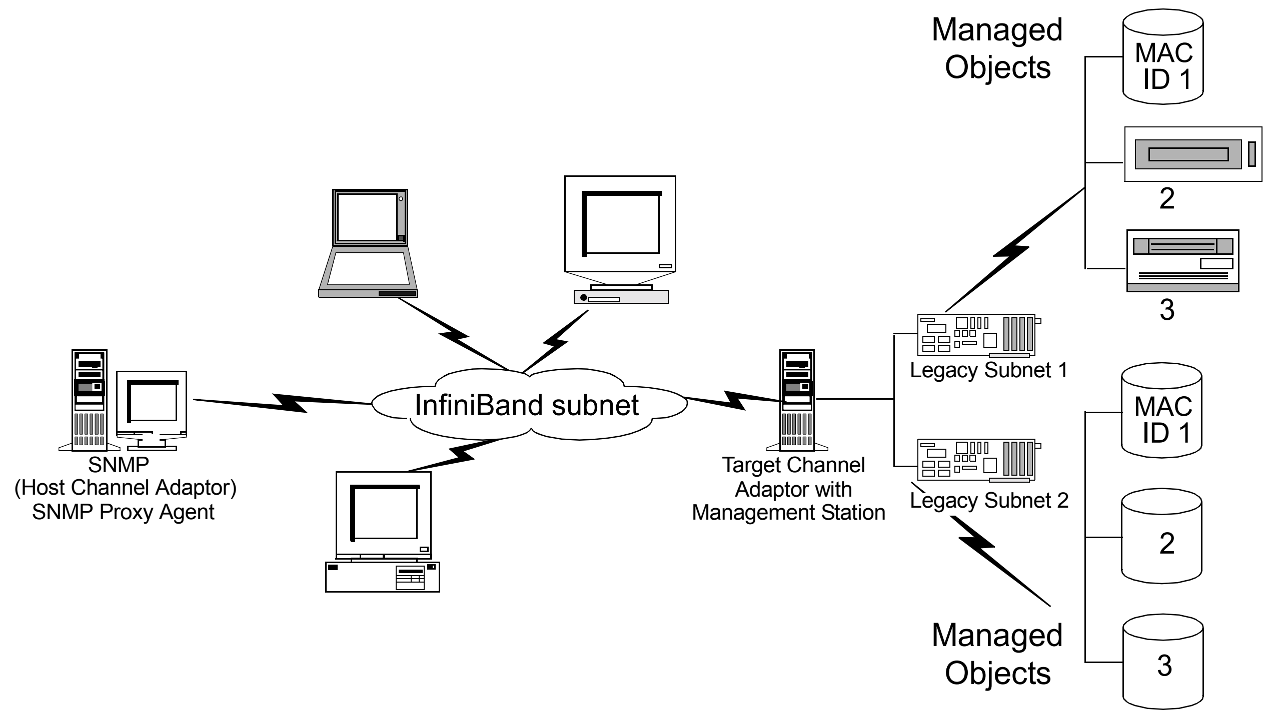


Figure 182 SNMP Proxy Agents

To target a remote managed object not directly connected to the InfiniBand fabric requires the use of an SNMP Proxy Agent. See Figure 182. The basic function of a Proxy Agent is to receive SNMP packets passed up from the InfiniBand Endnode SNMP agent and forward them to that remote managed agent. These remote agents are, as such, not directly connected to the InfiniBand fabric and thus cannot be managed through it unless an intermediate device acts on its behalf to receive and send over the unsupported interconnect.

o16-11: The InfiniBand architecture shall be able to accommodate such legacy transports by redirecting SNMP packets destined for these managed nodes.

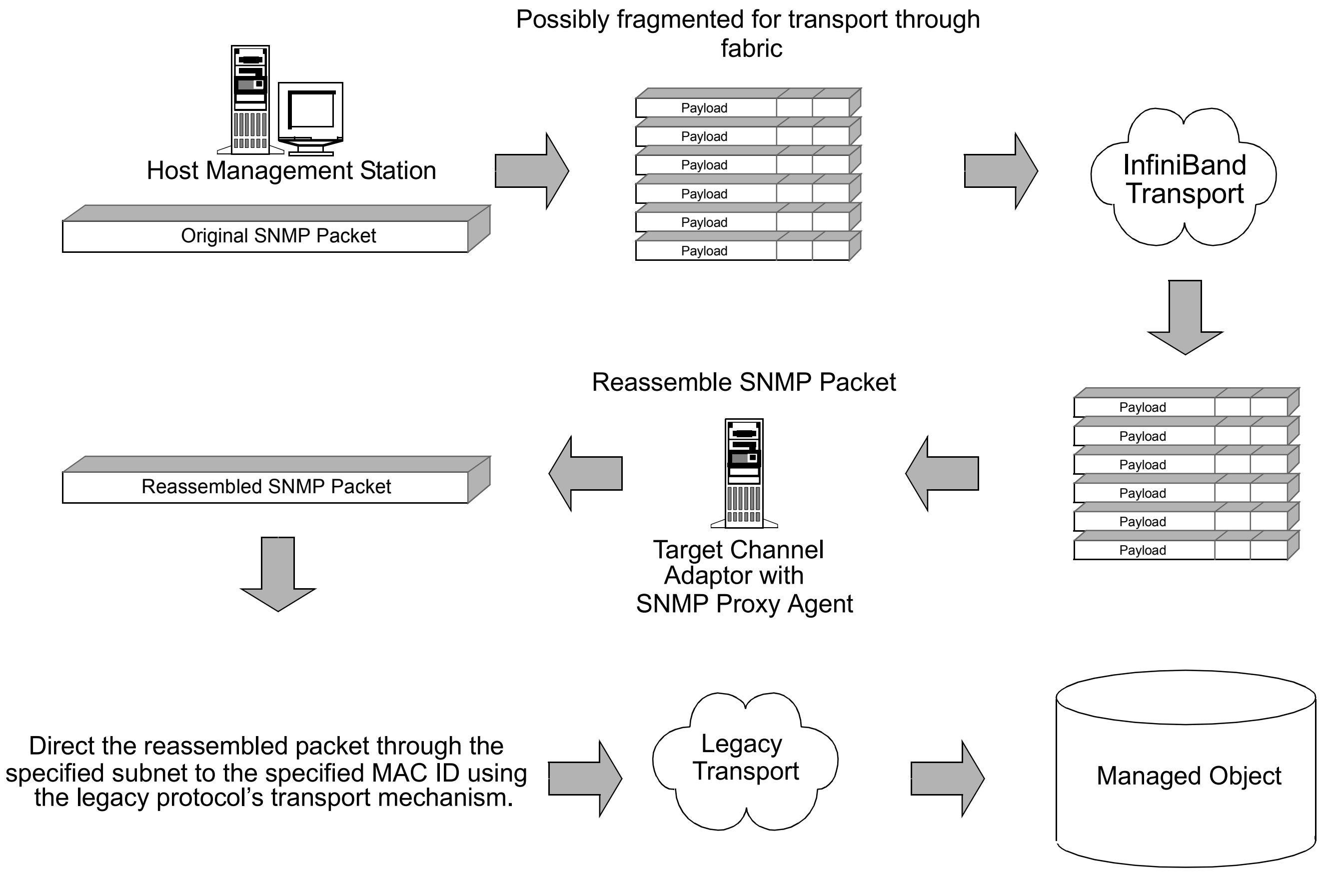


Figure 183 SNMP PDU Segmentation

SNMP targeting for beyond the InfiniBand Endnode (such as an InfiniBand device attached to a TCA that supports SNMP) is accomplished by an SNMP redirect. An SNMP packet destined for such a redirection will contain one of the SNMP redirect features and specify the destination address in the *raddress* field of the SNMP class datagram. This will allow the Proxy SNMP Agent to reassemble the SNMP packet from its fragments (if any) so it may re-encode the packet for the legacy transport over which it travels to reach its final destination.

16.4.4.2 Trap Event Subscription

A node may request SNMP traps from a given node be sent to it. This is done by setting the ClassPortInfo Attribute with the LID and QP appropriately. The SNMP agent will transmit the Trap PDU as a sequence of

SNMP datagrams to the destination node, that is it is not using the method Trap() but the method Send() with the Trap PDU as the SNMP Data.

16.5 VENDOR-SPECIFIC

The Vendor-specific Agent is optional.

Vendor-specific operations can be defined using the vendor-specific management classes.

Vendors are free to define new methods and attributes and their use, provided that they conform to the common MAD format and methods defined herein, and do not conflict with the stated restrictions on method and attribute utilization.

16.5.1 MAD FORMAT

o16-12: The datagrams in these Vendor-specific classes shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 184 Vendor MAD Format on page 813 andTable 237 Vendor MAD Fields on page 813 below.

Figure 184 Vendor MAD Format

bytes	
0	Common MAD Header
...	
20	
24	Data
...	
252	

Table 237 Vendor MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 604
Data	232 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.5.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. Class-specific bits are defined by the Vendor.

Table 238 Vendor Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617
8-15	-	Class-specific bits defined by Vendor

16.5.2 METHODS

Vendor classes supports the common methods

Table 239 Vendor Class Methods

Method Type	Value	Description
VendorGet()	0x01	Request an attribute to return (read) from a target.
VendorSet()	0x02	Request a target to set (write) an attribute. The object will issue a VendorGetResp() as a response.
VendorGetResp()	0x81	Target responds to an attribute Get/Set request.
VendorSend()	0x03	Send a datagram. Does not require a response.
VendorTrap()	0x05	Unsolicited datagram sent to the vendor entity. Contains the Notice Attribute as defined in 13.4.8.2 Notice on page 622 to identify the trap.
VendorTrapRepress	0x07	Block repetition of notification.

Vendor-specific methods can be added as desired by vendor, providing there is no collision with reserved methods in 13.4.5 Management Class Methods on page 607.

16.5.3 ATTRIBUTES

o16-13: The Vendor classes shall support the ClassPortInfo attribute.

The Vendor classes may optionally support the attributes Notice and InformInfo. All other attributes are vendor-defined.

Table 240 Vendor Class Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x00000000	See 13.4.8.1 ClassPortInfo on page 619
Vendor defined	0x0010 - 0xFFFF	0x00000000-0xFFFFFFFF	

Table 241 Vendor Attribute / Method Map

Attribute Name	VendorGet	VendorSet	VendorSend	VendorTrap
ClassPortInfo	x	x		
Vendor defined	Vendor defined			

16.5.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. Class-specific bits are defined by Vendor.

Table 242 Vendor ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619
8-15	-	Class-specific bits defined by Vendor

16.6 APPLICATION-SPECIFIC

The Application-specific Agents are optional.

Application-specific operations can be defined using the application-specific management classes.

Applications are free to define new methods and attributes and their use, provided that they conform to the common MAD format and methods defined herein, and do not conflict with the stated restrictions on method and attribute utilization.

16.6.1 MAD FORMAT

o16-14: The datagrams in these Application-specific classes shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 185 Application MAD

Format on page 816 andTable 243 Application MAD Fields on page 816 below.

Figure 185 Application MAD Format

bytes	
0	Common MAD Header
...	
20	
24	Data
...	
252	

Table 243 Application MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 604
Data	232 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.6.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. Class-specific bits are defined by the Application.

Table 244 Application Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617
8-15	-	Class-specific bits defined by Application

16.6.2 METHODS

Application classes supports the common methods

Table 245 Application Class Methods

Method Type	Value	Description
AppGet()	0x01	Request an attribute to return (read) from a target.
AppSet()	0x02	Request a target to set (write) an attribute. The object will issue a AppGetResp() as a response.
AppGetResp()	0x81	Target responds to an attribute Get/Set request.

Table 245 Application Class Methods

Method Type	Value	Description
AppSend()	0x03	Send a datagram. Does not require a response.
AppTrap()	0x05	Unsolicited datagram sent to the application entity. Contains the Notice Attribute as defined in 13.4.8.2 Notice on page 622 to identify the trap.
AppTrapRepress	0x07	Block repetition of notification.

Application-specific methods can be added as desired by application, providing there is no collision with reserved methods in 13.4.5 Management Class Methods on page 607.

16.6.3 ATTRIBUTES

o16-15: The Application classes shall support the ClassPortInfo attribute.

The Application classes may optionally support the attributes Notice and InformInfo. All other attributes are application-defined.

Table 246 Application Class Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x00000000	See 13.4.8.1 ClassPortInfo on page 619
Application defined	0x0011 - 0xFFFF	0x00000000-0xFFFFFFFF	

Table 247 Application Attribute / Method Map

Attribute Name	AppGet	AppSet	AppSend	AppTrap
ClassPortInfo	x	x		
Application defined	Application defined			

16.6.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. Class-specific bits are defined by Application.

Table 248 Application ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619
8-15	-	Class-specific bits defined by Application

16.7 COMMUNICATION MANAGEMENT

C16-14: This compliance statement is now obsolete.

Communication Management is described in Chapter 12: Communication Management on page 545. The Communication Management functions required for nodes are described in that chapter. Proper use of the messages defined in this section is subject to the protocols and state machines defined in that chapter. No semantics is described in this section.

16.7.1 MAD FORMAT

C16-15: The datagrams in the Communication Management class shall conform to the MAD format and use as specified in 13.4 Management Datagrams on page 603 and further customized in Figure 186 Communication Management MAD Format on page 818 and Table 249 Communication Management MAD Fields on page 818 below.

Figure 186 Communication Management MAD Format

bytes	
0	Common MAD Header
...	
20	
24	Data
...	
252	

Table 249 Communication Management MAD Fields

Field Name	Length	Description
Common MAD Header	24 bytes	Common MAD Header as described in 13.4.2 Management Datagram Format on page 604
Data	232 bytes	Attribute data is mapped bit for bit from the format described in the following sections to the start of this data field. If the attribute is smaller than the data field, the content of the remainder of the data field is unspecified.

16.7.1.1 STATUS FIELD

The Status field is described in 13.4.7 Status Field on page 617. No class-specific bits are defined.

Table 250 Communication Management Status Field

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.7 Status Field on page 617

Table 250 Communication Management Status Field

Bits	Name	Meaning
8-15	-	Class-specific bits are reserved

16.7.2 METHODS

The Communication Management class supports the methods identified in Table 251 Communication Management Methods on page 819 below.

Table 251 Communication Management Methods

Method Type	Value	Description
ComMgtGet()	0x01	Request a get (read) of an attribute
ComMgtSet()	0x02	Request a set (write) of an attribute.
ComMgtGetResp()	0x81	Response from a get or set request.
ComMgtSend()	0x03	Send a connection management message.

16.7.3 ATTRIBUTES

The Attributes/Attribute Modifiers specified in this section describe the mappings of message parameters defined section 12.4 Communications Services into the standard MAD header/payload format. The set of attributes supported by the Communication Management class is listed in Table 252 Communication Management Attributes on page 819.

Table 252 Communication Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
ClassPortInfo	0x0001	0x00000000	Refer to 13.4.8.1 ClassPortInfo on page 619
ConnectRequest	0x0010	0x00000000	Refer to 12.6.5 REQ - Request for Communication on page 551
MsgRcptAck	0x0011	0x00000000	Refer to 12.6.6 MRA - Message Receipt Acknowledgment on page 553
ConnectReject	0x0012	0x00000000	Refer to 12.6.7 REJ - Reject on page 554
ConnectReply	0x0013	0x00000000	Refer to 12.6.8 REP - Reply to Request for Communication on page 558
ReadyToUse	0x0014	0x00000000	Refer to 12.6.9 RTU - Ready To Use on page 559
DisconnectRequest	0x0015	0x00000000	Refer to 12.6.10 DREQ - Request for communication Release (Disconnection REQuest) on page 559

Table 252 Communication Management Attributes

Attribute Name	Attribute ID	Attribute Modifier	Description
DisconnectReply	0x0016	0x00000000	Refer to 12.6.11 DREP - Reply to Request for communication Release on page 559
ServiceIDResReq	0x0017	0x00000000	Refer to 12.6.5 REQ - Request for Communication on page 551
ServiceIDResReqResp	0x0018	0x00000000	Refer to 12.6.8 REP - Reply to Request for Communication on page 558
LoadAlternatePath	0x0019	0x00000000	Refer to 12.8.1 LAP - Load Alternate Path on page 570
AlternatePathResponse	0x001A	0x00000000	Refer to 12.8.2 APR - Alternate Path Response on page 571

Table 253 Communication Management Attribute / Method Map on page 820 indicates the methods with which each of the attributes is valid.

Table 253 Communication Management Attribute / Method Map

Attribute	ComMgtGet	ComMgtSet	ComMgtSend
ClassPortInfo	x	x	
ConnectRequest			x
ConnectReply			x
ReadyToUse			x
MsgRcptAck			x
ConnectReject			x
DisconnectRequest			x
DisconnectReply			x
LoadAlternatePath			x
AlternatePathResponse			x
ServiceIDResReq			x
ServiceIDResReqResp			x

The normative definitions of the attribute components and the operational requirements and constraints applicable thereto are defined in.

16.7.3.1 CLASSPORTINFO

The ClassPortInfo attribute is described in 13.4.8.1 ClassPortInfo on page 619. In addition, bit 8 through 12 of the CapabilityMask component are defined:

Table 254 Communication Management ClassPortInfo:CapabilityMask

Bits	Name	Meaning
0-7	-	Common bits as defined in 13.4.8.1 ClassPortInfo on page 619
8	IsMulticastCapable	Multicast is supported
9	IsReliableConnectionCapable	Reliable Connections are supported
10	IsReliableDatagramCapable	Reliable Datagrams are supported
11	IsRawDatagramCapable	Raw Datagrams are supported
12-15	-	Reserved

CHAPTER 17: CHANNEL ADAPTERS

17.1 OVERVIEW

This section specifies the minimum requirements for an IBA channel adapter. Channel adapters (CA) are the source and terminus of IBA packets that traverse the IBA switching fabric. Channel adapters are either Host Channel Adapters (HCAs) or Target Channel Adapters (TCAs). In a typical system, the HCAs are used by the host processors to connect to the IBA fabric whereas the TCAs are used by an I/O adapter to connect to the IBA fabric.

The key difference between a Host Channel Adapter and a Target Channel Adapter is in the way the client (whether the client is hardware or software) interfaces to the transport layer. Specifically, the HCA supports the IBA Verbs layer whereas the TCA uses an implementation dependent interface to the transport layer.

C17-1: An HCA shall support the IBA verbs layer interface.

Previous sections of the specification have described the various layers comprising an IBA Channel Adapter (physical, link, network, and transport layers). All channel adapters share a common architecture for the physical, link, network and transport layers. See Figure 187 below. From the point of view of the physical communications link, an HCA and TCA are identical.

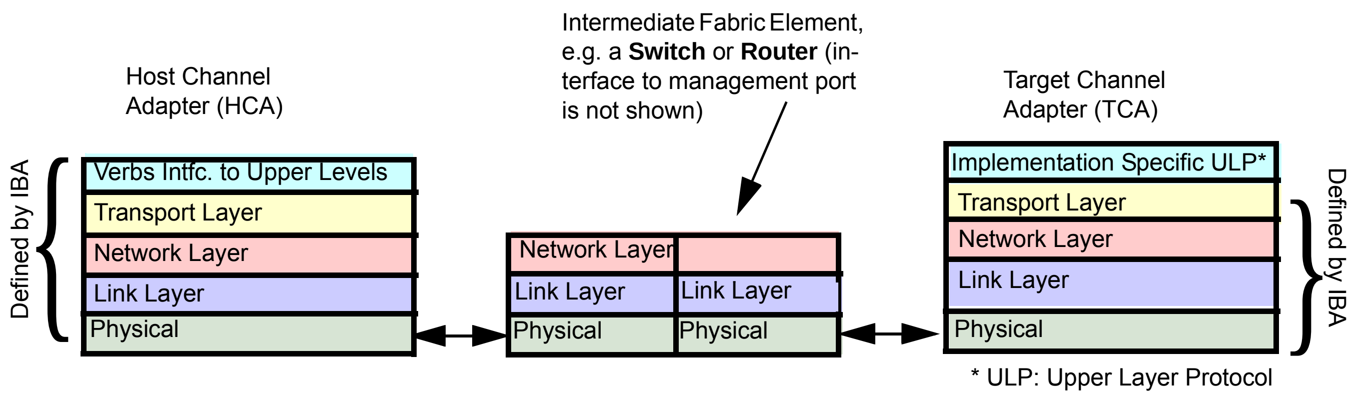


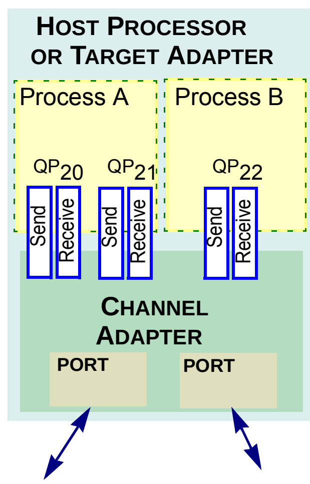
Figure 187 IBA Architecture Layers

This chapter lists the common functionality in all CAs as well as the differences between HCAs and TCAs. There are also differences in required minimum functionality. These issues are addressed in the following sections.

17.2 Common Functional Requirements

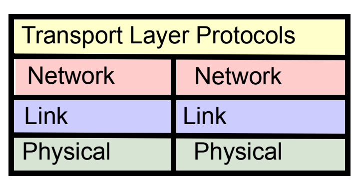
17.2.1 Multiple Ports per Channel Adapter

A Channel Adapter[1] may have one or more ports. A CA's port provides the physical, link and network protocol layers of the IBA CA. A channel adapter with multiple ports shares the transport layer functionality amongst the ports. For example, a QP (a transport layer construct) can be configured to work with any of the ports on the CA. The following figures show the physical representation as well as the architectural layering.



A two ported Channel Adapter. The CA has one block of QPs, memory translation tables etc. that interact with the IBA fabric through the two ports.

Figure 188 Multiport CA



A two ported CA showing a common transport layer and independent Network, Link and Physical Layers for each port.

Figure 189 Multiport CA Architectural Layers

It is desirable for a channel adapter to have multiple ports for several reasons:

---

1. Unless specifically mentioned, the term Channel Adapter refers to both an HCA and a TCA.

- **Increased bandwidth from a single CA**. e.g. an HCA with a high performance host memory interface can support the bandwidth of several IBA links. By adding relatively low cost IBA ports the HCA can multiply its throughput with relatively little additional cost. See Figure 191 on page 825.
- **Redundant paths for fault tolerant communications**. In a system with multiple paths between source and destination, a CA's multiple ports may be used to tolerate faults in the fabric's switches and links. See Figure 192 on page 825 and Figure 193 on page 826.
- **Support direct links to TCAs.** in a low cost, switchless topology the ports of an HCA might be directly wired to TCAs. See Figure 194 on page 826.

17.2.1.1 TOPOLOGIES SUPPORTED WITH MULTI-PORTED CHANNEL ADAPTERS

The following diagrams show several basic ways a multi ported CA could be attached to the rest of the system. These diagrams are by no means the only topologies supported -- they are examples only. Note that multiple ports on a CA may either connect to multiple subnets or to the same subnet.

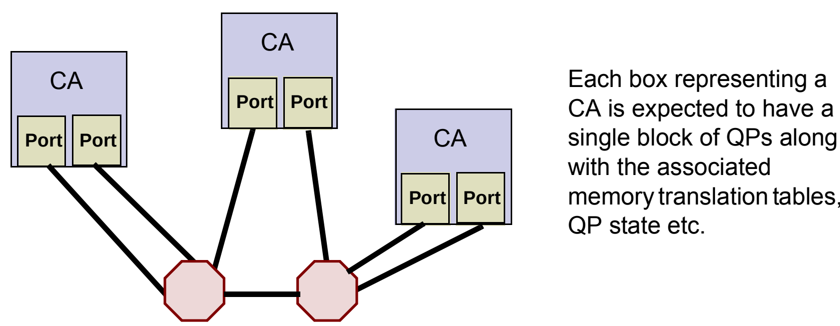


Figure 190 Multiported CAs Connected to Single Subnet

C17-2: A multiported CA shall be capable of connecting to one or more subnets.

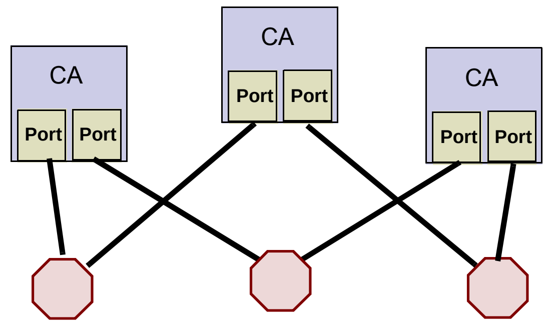


Figure 191 Multiported CAs Connected to Multiple Subnets

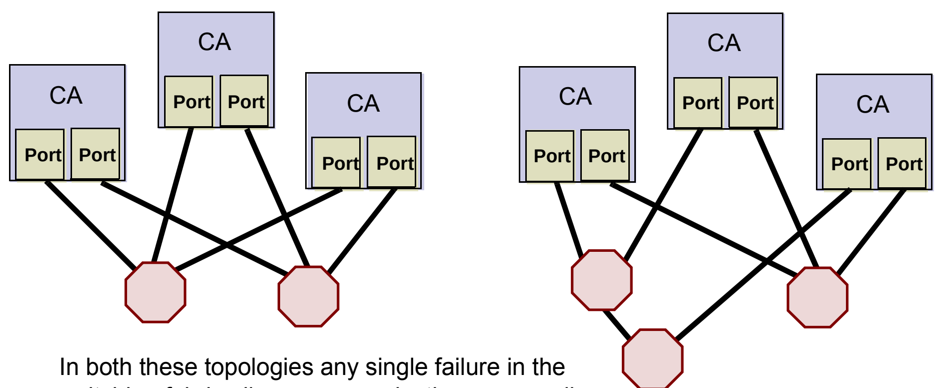


Figure 192 Fault Tolerant Connections to Independent Fabrics

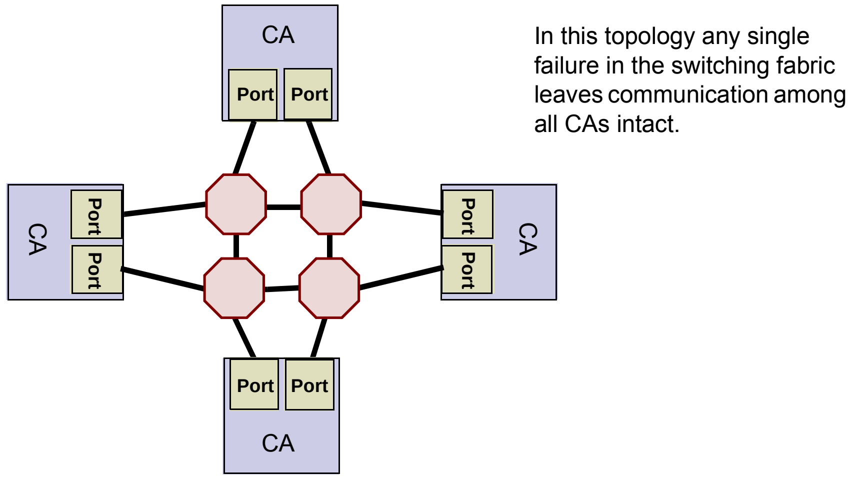


Figure 193 Fault Tolerant Connection to a Single Fabrics

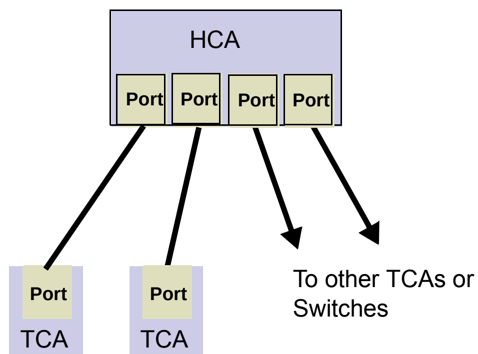


Figure 194 Multiported HCA with Direct Connections to TCAs

17.2.1.2 ASSOCIATION OF QPS WITH PORTS

C17-3: While a CA may have many QPs and many ports, each QP shall generate request packets, service returning response packets, and respond to arriving request packets through exactly one port, at any point in time.

To ensure packet ordering within a QP for connected or reliable transport services, packets are required to take the same path between a source and destination. This requires that all requests and responses use a consistent port, base SLID, DLID, VL and SL. A connected or reliable transport QP remains bound with one port until path migration for error recovery or load balancing purposes occurs or the connection goes away.

Aside from valid packets requesting path migration as described in Section 17.2.8 Automatic Path Migration on page 836, incoming request and response packets arriving at a port other than the port currently bound to the appropriate QP may be discarded.

For an HCA using the unreliable datagram transport service, the verbs layer specifies the remote address with each outgoing work request. Since the QP is only bound to one port, the client of the verbs layer must be certain the destination is reachable from that port. In certain topologies not all destinations are reachable from all ports (see Figure 191 on page 825).

Incoming Unreliable Datagram packets may only target a QP if that QP is bound to the port on which the datagram arrived.

The Reliable Datagram service uses an end-to-end context to ensure correct delivery for every channel adapter with which it communicates. The EE context, like a Reliable Connection QP, is bound to one port (at least until path migration is used to rebind the EE context with a new port). But since a RD QP can communicate with multiple EE contexts, the RD QP can in effect be transmitting and receiving packets from multiple ports.

17.2.1.3 PORT ATTRIBUTES AND FUNCTIONS

Certain attributes and functions are associated with each port. Typically these belong to the physical, link, and network layers that are unique to the port. The table below itemizes these as well as describes some transport layer functionality unique to each port. Each attribute or function is intended to be applied individually to each port.

Table 255 Port Attributes & Functions

Attribute/Function	HCA	TCA
Physical Interface	The HCA and TCA shall support one or more of the IBA defined physical interfaces. (See the IBA Specification, Volume 2)	
Static Rate Control (limiting the BW to a particular destination CA)	required on ports supporting bandwidths above 2.5 Gbps	
Support for multipathing (see Section 7.11 Subnet Multipathing on page 192)	required	required

Table 255 Port Attributes & Functions

Attribute/Function	HCA	TCA
P_Key Checking on inbound Request and Inbound Response Packets (see Section 10.9 Partitioning on page 454)	Shall validate incoming packet's P_Key with the P_Key bound to the destination QP[a]. A CA shall maintain a P_Key table (see Section 10.9.2 The Partition Key Table (P_Key Table) on page 456) per port. Each table shall have at least one P_Key entry. HCA requires no OS involvement to set the P_Key (i.e. P_Key is set directly by a Subnet Manager control packet.)	
Validation of incoming packet's DLID and, if the GRH is present, DGID	required	required
Support for QP0 and QP1	required on each port	required on each port
Port Numbering	Ports are numbered starting from one and if there are multiple ports, they are numbered sequentially. MADs use port number zero as a wild-card port number that matches whatever port the packet arrived at. See 14.2.5.6 PortInfo on page 665.	
GID Support	Each port has at least one GID. The maximum number of GIDs per port is implementation specific. See the discussion on GIDs in Section 4.1 Terminology And Concepts on page 116.	

a. excluding Raw Datagram QPs (because raw datagrams don't have a P_Key). Also PKey checking for QP0 and QP1 are slightly different. See Section 10.9.8 Partition Enforcement on Management Queue Pairs on page 460 for more information.

C17-4: Static rate controls, as listed in Section 17.2.6 Static Rate Control on page 835, are required on each port that supports a data rate above 2.5 gbps.

C17-5: Each port shall validate the incoming P_Key in an IB Transport packet with the P_Key bound to the destination QP (other than QP0 and QP1).

C17-6: The CA shall maintain a P_Key table per port supporting at least one and at most 65,535 P_Key entries.

C17-7: An HCA shall require no OS involvement to set the P_Key table; the P_Key table shall be set directly by Subnet Manager MADs.

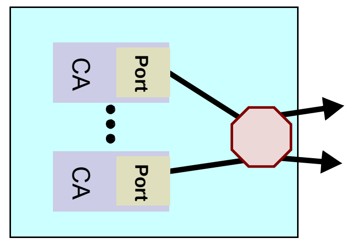
C17-7.a1: A CA may support up to 254 ports. For a CA supporting N ports, the ports shall be numbered from 1 to N.

C17-8: Each port shall support at least one GID.

17.2.1.4 SWITCHING PACKETS THROUGH MULTIPLE PORTS

If a Channel Adapter has multiple ports, the CA does not route packets from one port to the other. Such a packet forwarding function is defined as a switch.

An implementation may choose to package a switch and multiple IBA ports together, as shown in the figure below.



The overall box shows the boundary of the combined switch and Channel Adapter. This boundary could be a single IC or board. The boundary also represents the fault zone for that device.

Figure 195 Multiple Single Ported CAs with an Embedded Switch

17.2.2 CHANNEL ADAPTER ATTRIBUTES

The previous section described attributes of a channel adapter's ports. This section describes attributes of the whole channel adapter.

This specification only sets the minimum functionality of an HCA or TCA. For example, only two QPs are required (both for management). A practical HCA or TCA would undoubtedly support more QPs, but this section only specifies architectural minimum requirements. The following summarizes various required and optional Channel Adapters attributes (see also section 11.2.1.2 Query HCA on page 476).

Table 256 Channel Adapter Attributes

Attribute	HCA	TCA
Support for multiple ports.	Optional	Optional
Source/Sink packets with a LRH (for communication within the subnet)	Required for all QPs.	
Source/Sink packets with a GRH (for communication across subnets)	Required for all QPs other than QP0.	
Transport Services Supported	HCAs shall be capable of supporting the Unreliable Datagram, Reliable Connection, and Unreliable Connection transport service on any QP supported by the HCA.	Aside from supporting Unreliable Datagram for the two required management QPs, support for any other transport service (or QP) is optional.

Table 256 Channel Adapter Attributes

Attribute	HCA	TCA
Atomic Operations Supported	Optional to generate requests or responses	
Other Operations Supported	If a transport service is supported, then the CA must support all the operations defined for that transport service (excluding atomic operations).	TCAs are not general purpose and may customize the operations supported to suit their function (e.g. a TCA with Reliable Connection Service may generate RDMA requests but not respond to incoming RDMA requests)
Solicited Events (see Section 9.2.3 Solicited Event (SE) - 1 bit on page 209 and Section 11.4.2.2 Request Completion Notification on page 535	Required to both generate solicited events and to receive them.	Optional
Path MTU	CAs shall support one of the following sets of MTUs (for all Transport Service Classes): • 256 Bytes • 256, 512 Bytes • 256, 512, 1024 Bytes • 256, 512, 1024, 2048 Bytes • 256, 512, 1024, 2048, 4096 Bytes	
	For UD and Raw the WQE determines the MTU. For RD the EE context specifies the MTU. For RC and UC the QP specifies the MTU.	Selection of MTU for TCAs is implementation specific.
End-to-End Flow Control (reliable connection transport service only)	HCA receive queues shall generate E-to-E flow control credits • i.e. HCAs throttle inbound requests to prevent inbound Sends arriving at an empty receive queue. HCA send queue shall receive and respond to inbound credits • i.e. remote node may throttle the HCA's outbound requests.	TCA receive queues may generate E-to-E flow control credits. • i.e. TCA need not throttle inbound requests. TCA send queue shall receive and respond to inbound credits. • i.e. remote node may throttle the TCA's outbound requests.
Multicast	Generating IBA Raw Multicast packets is optional. Receiving IBA Raw Multicast packets is optional. Generating IBA Unreliable Datagram Multicast packets is optional[a]. Receiving IBA Unreliable Datagram Multicast packets is optional.	
Automatic Path Migration	It is optional to either generate or respond to an automatic path migration request.	
Memory Protection	HCA's provide memory protection as described in Section 10.6 Memory Management on page 427.	Optional

Table 256 Channel Adapter Attributes

Attribute	HCA	TCA
Loopback Support	Self addressed packets[b] shall be allowed and shall not go out onto the wire. That is, self addressed packets must work even if no external switch is present	Optional

a. It is expected that any implementation of the Unreliable Datagram transport service will trivially support the generation of multicast packets.

b. A self-addressed packet is a packet whose DLID and SLID (while not necessarily identical) address the same CA. A self-addressed packet may or may not have the same source and destination QP. IB does not define a specific “loopback” address.

C17-9: All channel adapters shall be able to source and sink (to all QPs) locally routed packets (i.e. no GRH).

C17-10: All channel adapters shall be able to source and sink (to all QPs other than QP0) globally routed packets (i.e. packets with a GRH).

C17-11: HCAs shall be capable of supporting the Unreliable Datagram, Reliable Connection, and Unreliable Connection transport service on any QP supported by the HCA.

C17-12: If a transport service is supported by an HCA, then that HCA must support all the operations defined for that transport service (excluding atomic operations).

C17-13: An HCA shall be able to generate and receive solicited event.

C17-14: CAs shall support one of the following sets of MTUs (for all Transport Service Classes):
256 Bytes
256, 512 Bytes
256, 512, 1024 Bytes
256, 512, 1024, 2048 Bytes
256, 512, 1024, 2048, 4096 Bytes

C17-15: HCA receive queues shall generate E-to-E flow control credits.

C17-16: HCA send queue shall receive and respond to inbound E-to-E flow control credits.

o17-1: TCA receive queues may generate E-to-E flow control credits.

C17-17: TCA send queue shall receive and respond to inbound E-to-E flow control credits.

o17-2: A CA may be capable of generating multicast packets.

o17-3: A CA may be capable of receiving multicast packets.

o17-4: A CA may be capable of generating and responding to the Automatic Path Migration protocol.

C17-18: HCAs shall allow packets with a destination address the same as that of the port on which the packet is issued. Such a loopback packet shall not go onto the wire.

17.2.3 DEADLOCK PREVENTION

Each CA shall not cause deadlock in the fabric. This condition is met by

- The CA will not continuously and permanently assert backpressure (i.e. fail to grant link credits).
- The CA shall not assert backpressure on a port's inbound link as the result of receiving backpressure on that port's outbound link.

C17-19: For deadlock prevention, the CA shall not continuously and permanently assert backpressure (i.e. fail to grant link credits).

C17-20: For deadlock prevention, the CA shall not assert backpressure on a port's inbound link as the result of receiving backpressure on that port's outbound link.

17.2.4 CHECKING INCOMING PACKETS

All CA's are required to validate each incoming packet before committing the packet to the CA's state.

C17-21: The CA shall check for link, network and transport layer errors in all incoming packets.

17.2.5 NON-VOLATILE STATE

C17-22: All channel adapters shall maintain a EUI-64 port GUID and a EUI-64 CA GUID (See Chapter 4: Addressing on page 115) in nonvolatile memory such that the GUID is the same each time the CA is powered on.

C17-23: Any CA that can become a Subnet Manager (see Section 14.2 Subnet Management Class on page 642) shall also keep its Subnet Manger Priority in nonvolatile memory.

Other uses of the nonvolatile memory are optional.

The type of non volatile memory in a CA is not specified and might be a local disk drive or on-chip memory.

IBA does not require a CA to remember connection state information across power cycles.

17.2.6 STATIC RATE CONTROL

A CA shall support static rate control (see section 9.11 Static Rate Control on page 393) if its raw bandwidth is greater than 2.5 Gbps. The Inter-packet Delay (IPD) values supported (see Table 63 on page 394) must allow slowing the packet rate to all of the standard link rates. The table below indicates the values of IPD that shall be supported

Table 257 Static Rate Control IPD Values

IPD	rate	Comment
0	100%	Required by all CAs
3	25%	Required by CAs that support 1 GB/s or higher link rate
2	33%	Required by CAs that support 3 GB/s or higher link rate
11	8%	Required by CAs that support 3GB/s or higher link rate

17.2.7 MANAGEMENT MESSAGES

Each port of every channel adapter shall support two QPs for management commands:

- QP0, used by the Subnet Management Agent for sending and receiving Subnet Management Packets (SMPs).
  - This QP uses the Unreliable Datagram transport service.
  - SMP packets arriving before the current packet's command completes may be dropped (i.e. the minimum queue depth of QP0 is one).
- QP1, used for the General Services Interface (GSI).
  - This QP uses the Unreliable Datagram transport service.
  - All traffic to and from this QP uses any VL other than VL15.
  - GSI packets arriving before the current packet's command completes may be dropped (i.e. the minimum queue depth of QP1 is one).

C17-24: Each port of every CA shall support QP0 for use by the SMA and QP1 for use by the GSA.

All QPs for a given CA, except QP0 and QP1 have unique numbers. QP0 and QP1 are special in that each port has its own QP0 and QP1.

The rest of the (non RD) QPs on a CA may be bound with any one port. The binding of a QP (other than QP0 or QP1) with a port is maintained until such time that automatic path migration (see 17.2.8 Automatic Path

Migration on page 836) or path migration requested by MADs associates the QP with a different port.

The management QPs are special because they are used by the Subnet Manager and other management applications. See Section 13.5.1 MAD Interfaces on page 630.

Since each port may be on a different subnet it must communicate with a different Subnet Manager and related management application, The Subnet Manager and other nodes using the GSI use the well known QP numbers (0 and 1) to establish communication.

17.2.7.1 SUBNET MANAGEMENT

All CAs shall respond to incoming Subnet Management Packets from the Subnet Manager. CAs shall also generate the required traps defined as part of the SMA.

The IBA does not require nor preclude a CA from being a Subnet Manager. If a node does host a Subnet Manager, it must meet the requirements as specified in section 14.4 Subnet Manager on page 687.

17.2.7.2 GENERAL SERVICES

All CAs shall respond to mandatory GSI MADs defined in Chapter 16: General Services on page 748. Any HCA or TCA may initiate MADs to another CA.

17.2.8 AUTOMATIC PATH MIGRATION

The reliable or connected transport services (Reliable Connection, Reliable Datagram, and Unreliable Connection) use the same path for a given connection (or in the case of RD for a given pair of end-to-end contexts). This ensures data is delivered in the proper order. Path migration refers to the requestor and responder agreeing to use a new path. The source and destination QPs remain the same but the ports and path through the fabric may change. Path migration may be used to recover from a bad path (sometimes this is referred to as Failover) or for other reasons such as load balancing.

Automatic path migration may be supported by HCAs and TCAs. If supported, Automatic path migration works for QPs using the RC, RD, and UC transport services.

Automatic path migration provides a fast mechanism for path migration. When a connection is established the two CA's use Communication Management MADs to establish the primary and alternate path (See sections 10.4 Automatic Path Migration on page 420 and 12.8 Alternate Path Management on page 569. Automatic path migration is a mechanism whereby either CA can signal the other to switch from the primary path to the alternate path.

At connection establishment time the CA is set with the following information to determine a path:

- DLID of the responding CA
- Destination GID of the responding CA
- SL
- source port (i.e. the base SLID and path bits for outbound request and response packets)

At connection establishment the CAs may be given two sets of path information, one for the primary path and another for the alternate path. The alternate path may use the same or different source and destination ports as that used by the primary path.

17.2.8.1 AUTOMATIC PATH MIGRATION PROTOCOL

The automatic path migration protocol uses a single bit in the BTH called MigReq (Migrate Request) and a 3-state state machine associated with each connected QP supporting automatic path migration. If automatic path migration is not supported by either QP of the connection, the state machines of the two QP's remain in the MIGRATED state.See figure below.

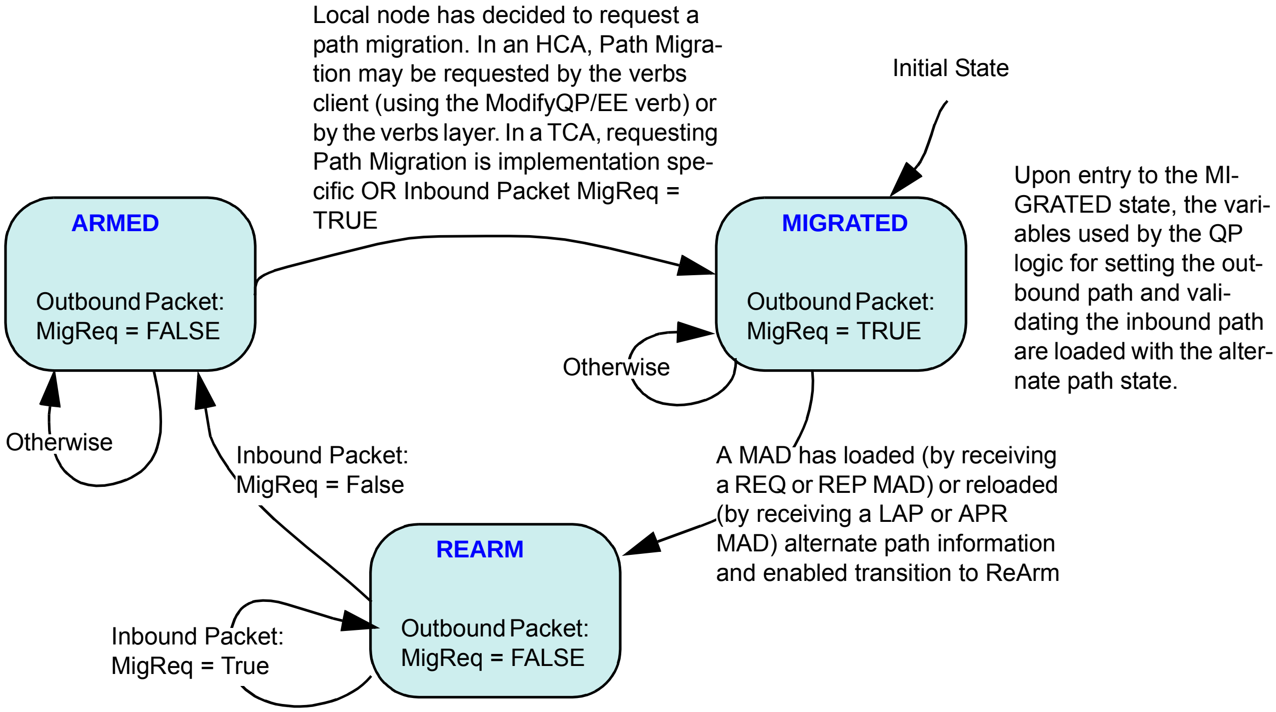


Figure 196 Automatic Path Migration State Machine (per QP)

17.2.8.1.1 INITIALIZATION

At connection setup time, the primary and alternate path states are assigned to each CA.

The Automatic Path Migration State Machine is initialized to the MIGRATED state whether or not Automatic Path Migration is supported by either QP of the connection.

17.2.8.1.2 MIGRATION REQUEST

Either CA may request an automatic path migration. Reasons for requesting automatic path migration are outside the scope of the IBA specification but may include load balancing or using an alternate path to recover from excessive errors.

The CA requesting automatic path migration transitions its state machine to the MIGRATED state. Once in the MIGRATED state the CA generates all new packets (both request and response packets) using the path that was previously initialized as the alternate path. The CA may refuse to accept incoming request or response packets arriving from the original path.

Once in the MIGRATED state, all outbound packets (both requests and responses) on that QP have the MigReq (Migrate Request) bit in the BTH set to TRUE.

17.2.8.1.3 MIGRATION RESPONSE

A CA whose QP is in the ARMED state that receives a packet (either request or response packet) with the MigReq bit set validates the incoming packets path bits with the alternate path information (i.e. checks the SLID, DLID, SGID, DGID against the alternate path state). If the validation passes the QP transitions to the MIGRATED state.

Upon entry to the MIGRATED state, the primary path used by the QP logic for setting the outbound path and validating the inbound path are loaded with the alternate path state. At this point all request and response packets from both CAs are using the alternate path.

17.2.8.1.4 RE-ENABLING MIGRATION

Migration is re-enabled via management intervention. First the alternate path variables are reloaded with new alternate paths. Then, based on a command from a management entity, the QP state is set to REARM. This causes the MigReq bit in outbound packets to be set false. Upon receiving an inbound packet with MigReq set false, the QP state is set to ARMED. Migration at this point is now re-enabled.

17.3 HOST CHANNEL ADAPTER

A HCA is differentiated from a TCA in that it supports the architecturally defined IBA Verbs Layer. As such, an HCA (and its vendor specific and OS specific driver SW) shall support the functionality of the Verbs Layer chapter.

17.3.1 LOOPBACK

An HCA shall be able to internally loopback a packet sent to itself. That is, the verbs layer can specify a packet to be delivered to the same port (possibly a different QP though). The packet shall be delivered without the packet appearing on the port's physical link. This loopback shall be able to function without requiring the presence of an external switch. Furthermore there is no special loopback address required.

On an HCA with multiple ports, a packet may be sent onto the wire from one port with the DLID in the packet targeting a different port. This is not considered loopback and follows all the normal rules for sending packets. An external switch is required for such a packet transfer, there is no requirement that a packet be routed internally from one port to another.

Loopback packets for diagnostic purposes that traverse an external switch are performed by using the directed routed subnet management packets.

17.4 TARGET CHANNEL ADAPTER

A channel adapter that attaches an I/O node to the fabric is a Target Channel Adapter (TCA). In most regards, a TCA is indistinguishable from an HCA when viewed from the perspective of the IBA wire semantics. However, there are certain characteristics and requirements that distinguish a TCA from an HCA. This section describes some of the differences between a target channel adapter and a host channel adapter, specifies functionality required of a TCA, and specifies minimum requirements on a TCA.

This section also describes the role of the target channel adapter in supporting its clients. Figure 197 illustrates the relationship of the target channel adapter in an I/O node. The client of the target channel adapter's services is one or more I/O controllers.

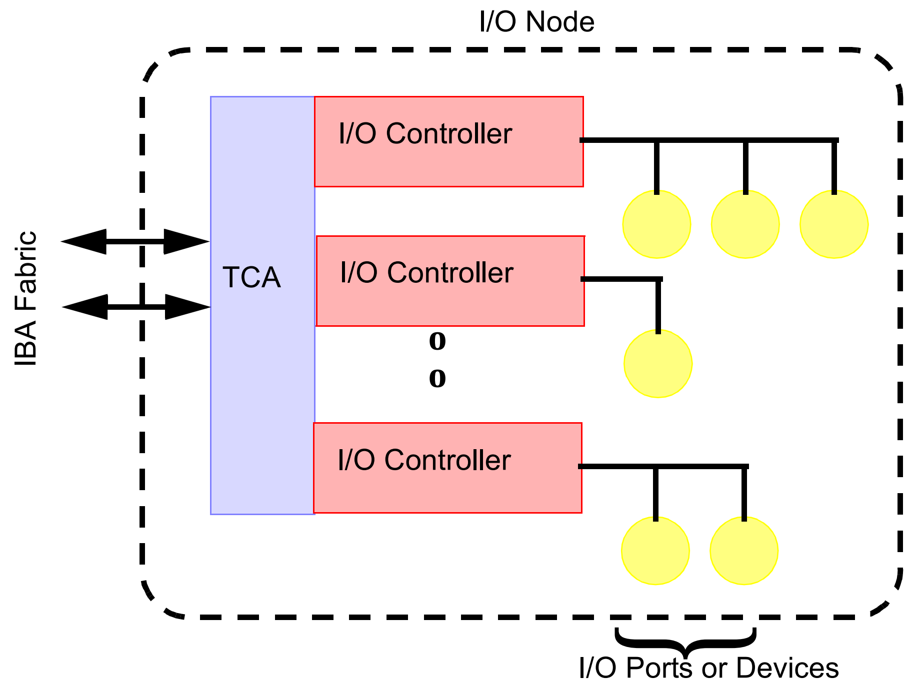


Figure 197 Generic I/O Node Model

17.4.1 CONTRAST TO A HOST CHANNEL ADAPTER

Unlike a host node, the execution environment for an I/O node is not necessarily associated with a general purpose processor. In fact, it can be entirely in hardware without any software environment.

For a host channel adapter, IBA specifies the semantics of the client interface characteristics (i.e., verbs) in order to support run time binding with the host's operating system and allow each component (HCA, OS, application) to be architected and distributed independently. But a target channel adapter can be bound to the I/O controller as part of the design process and distributed together. Thus the architecture does not specify any particular relationship between the target channel adapter and the I/O controller. This freedom promotes diversity and the ability to employ any queuing and notification mechanism that best serves the I/O function.

In the host environment, as illustrated, IBA service is separated into layers with the HCA hardware and the HCA driver being referred to as the host channel adapter. Thus the IBA services are not included in the requirements for the HCA. Instead, requirements for IBA services are applied to the host platform in general and not the HCA vendor.

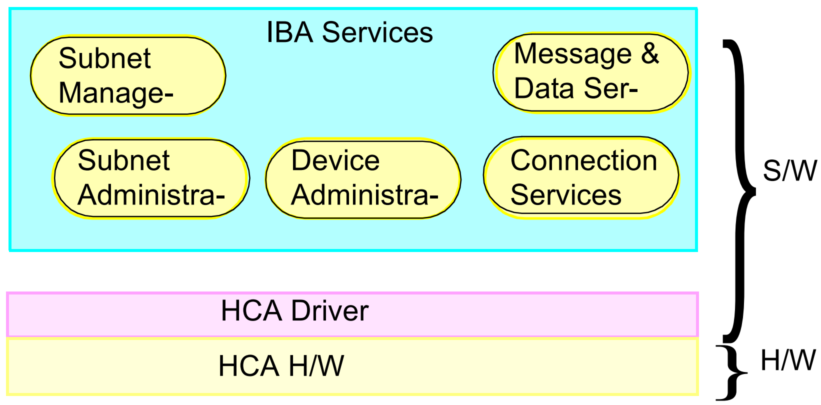


Figure 198 Host Environment - Split Responsibility

In the target environment, as illustrated, IBA services are not separated from TCA channel functionality, and thus the target channel adapter includes the IBA services. Thus the term target channel adapter is abstracted to mean all of the IBA mechanisms in the I/O node (target). TCAs may be implemented using software and hardware or hardware alone.

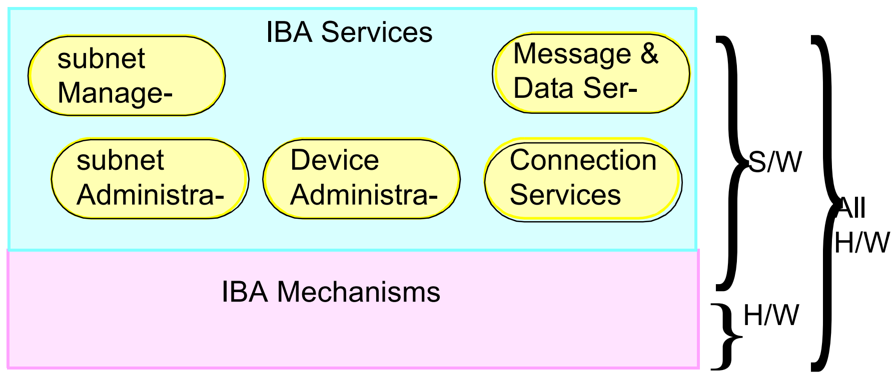


Figure 199 Target Environment - TCA Responsibility

A host channel adapter provides a generic service to its application “clients”. Therefore, IBA requires that an HCA provide full channel functionality. This is because the HCA vendor does not have prior knowledge of what applications will run over its channels.

However, since a target channel adapter vendor may have prior knowledge of the way the target channel adapter will be applied, the hardware vendor can reasonably restrict a TCA’s capabilities to only what is necessary for its clients.

17.4.1.1 MEMORY PROTECTION

IBA does not require that a TCA make any of its memory, or the memory associated with its attached I/O controllers directly accessible to another channel adapter. That is, there is no requirement that a TCA be capable of accepting inbound Atomic or RDMA READ or WRITE requests. If the TCA does expose its memory, or that of its attached I/O controllers, the architecture does not require that the TCA provide any form of memory protection, nor prescribe any particular mechanism for registering or protecting access to that memory other than the mechanism provided in the transport layer.

17.4.2 DEVICE ADMINISTRATION

Device administration packets allow an I/O node’s resources to be discovered and managed. In particular they provide the ability for a host to discover and invoke I/O services provided by I/O nodes.

A key distinction between a host and an I/O node lies in the method by which the I/O node's resources and capabilities are discovered, and the method by which connections to the TCA, and hence to the I/O resources on the node, are established. A complete set of messages is defined in Section 16.3 Device Management on page 793 for the purpose of allowing a consumer of the I/O node's services to discover the range of services offered by the I/O node.

Since a TCA is not required to support all IBA transport services, a particular TCA has associated with it a set of attributes defining its capabilities and the services it supports. Normally, these attributes are discovered by negotiation between peer channel adapters during the process of establishing a connection.

In the case of an I/O node which does not necessarily have the compute power and resources to participate in a complex negotiation, IBA defines a simple method by which a host or other intelligent I/O node can discover the target's attributes and establish connections accordingly.

Thus, IBA defines a rich set of I/O node attributes that can be read by an intelligent channel adapter and used during connection establishment in order to free the target from complex connection negotiation protocols. The host discovers target attributes directly, thus avoiding negotiation during connection establishment.

Each target must support the set of target/IO device attribute discovery messages as defined in Section 16.3.3 Attributes on page 797.

IBA does not specify the semantics nor methods between a TCA and its clients for conveying device information but it does specify the mechanisms and encoding for conveying that information.

IBA does not specify the semantics nor queueing models for the I/O controller to post messages, receive messages, and invoke RDMA Read, Write, and Atomic operations between a TCA and its clients.

IBA is I/O protocol agnostic. That is, how an I/O controller chooses to apply the services provided by the TCA is outside the scope of IBA as long as the usage corresponds to the rules for class of service and quality of service.

17.4.3 FABRIC LOOPBACK

A TCA does not have the same internal loopback requirement as does the HCA. Being a special purpose device, how the TCA handles packets addressed to itself is an implementation specific decision.

Loopback for diagnostic purposes that traverses an external switch is performed by using directed routed subnet management packets (just as done by the HCA)

CHAPTER 18: SWITCHES

18.1 OVERVIEW

This chapter specifies the requirements related to IBA switches.

Packets may be forwarded within a subnet (intra-subnet) and between subnets (inter-subnet). IBA switches are the fundamental forwarding component for intra-subnet routing (inter-subnet routing is provided by IBA routers, described later in this specification). Switches interconnect links by forwarding packets between the links.

Switches are transparent to the end stations and are not directly addressed (except for subnet management operations). To this end, every destination port within the network is configured with one or more unique Local Identifiers (LID's). From the point of view of a switch, a LID represents a path from the input port through the switch. Switch elements are configured with forwarding tables. Packets are addressed to their ultimate destination on the subnet using a destination LID (DLID), not to intervening switches. Individual packets are forwarded within a switch to an outbound port or ports based on the packet's DLID field and the Switch's forwarding table.

IBA switches are required to support unicast forwarding and may support multicast forwarding. In addition, IBA switches support a form of source routing, referred to as Directed Routing, for forwarding subnet management packets. This enables the configuration of a subnet without valid forwarding entries in the switches (e.g. a subnet power-up).

A Subnet Manager (SM) configures switches including loading their forwarding tables. The entity that communicates with the SM for the purpose of configuring the switch is referred to as the Subnet Management Agent (SMA). Every switch is required to have a subnet management agent. Individual switches within a power domain can be made observable to the SM via multiple instantiation of SMAs. Likewise, an SMA can be constructed that configures multiple switches and exports the multiple switches to the SM as a single switch; however, from the SM's perspective, such a configuration is a single switch.

Switches must also support a Subnet Management Interface (SMI) as specified in Chapter 14: Subnet Management on page 641 and a General Services Interface (GSI) as specified by Chapter 16: General Services on page 748. There are various mandatory and optional requirements of these interfaces that are specified in the respective chapters.

18.2 DETAILED FUNCTIONAL REQUIREMENTS

18.2.1 ATTRIBUTES

This section describes the major architecturally defined attributes of switches that are left as implementation choices.

Unicast Forwarding Table:

C18-1: For the forwarding of unicast packets, a switch shall implement either a linear forwarding table or a random forwarding table, but not both.

C18-2: A switch shall implement a unicast forwarding table with at least one entry and no more than 49,152 entries.

Two forms of an unicast forwarding table are defined: linear and random. All switches support one and only one of these forwarding table types. In either case, the required size for the unicast forwarding table is not specified by IBA and may vary between implementations. However, a valid range of table sizes is specified. Switches that implement the random form may also choose to limit the number of entries that may be assigned to a given port. This is further described in section 18.2.4.3 Packet Relay on page 851.

Multicast Support:

o18-1: The replication of multicast packets to multiple ports by switches is optional.

o18-2: A switch that implements the switch multicast replication service shall implement a multicast forwarding table with at least one entry and no more than 16383 entries.

IBA defines a switch multicast service that provides for the replication of packets by switches and their subsequent forwarding to multiple ports. The implementation of this service is optional. If implemented, IBA does not specify the size for the multicast forwarding table, and therefore the number of multicast groups a switch is capable of supporting. Consequently, the size of this table may vary by implementation. However, a valid range of table sizes is specified. Additional multicast requirements are specified in section 18.2.4.3.4 Optional Multicast Relay on page 857.

Virtual lanes:

C18-3: Switches shall implement the subnet virtual lane (also referred to as virtual lane 15).

o18-3: Switches may implement a single buffer resource shared by all ports for the subnet management virtual lane.

All switches implement the subnet management virtual lane (which is numbered virtual lane 15). Additionally, switches implement one, two, four, eight, or 15 data virtual lanes. These virtual lanes are numbered sequentially starting with zero. Unlike data virtual lanes, buffering for virtual lane 15 may be shared by all ports and may be shared by packet reception and transmission. This is described in 7.6 Virtual Lanes Mechanisms on page 153.

SL to VL mapping:

o18-4: Switches that implement more than one data virtual lane shall implement the SL to VL mapping function specified in this chapter.

o18-5: Switches that implement one data virtual lane may implement the SL to VL mapping function specified in this chapter.

SL to VL mapping is required on switches that support more than one virtual lane in addition to virtual lane 15. It is optional on switches that support only one virtual lane in addition to virtual lane 15. The specific requirements of this table are described in section 7.6.6 VL Mapping Within a Subnet on page 159.

P_Key Enforcement:

o18-6: Switches may implement the Inbound P_Key Enforcement Service specified in this chapter.

o18-7: Switches may implement the Outbound P_Key Enforcement Service specified in this chapter.

Switches may enforce partitions on ingress to and/or egress from the switch. This mechanism is described in sections 18.2.4.2.1 Inbound P_Key Enforcement on page 850 and 18.2.4.4.1 Outbound P_Key Enforcement on page 858.

Maximum Transfer Unit (MTU) size:

C18-4: Switches shall be capable of forwarding packets of size from the minimum valid packet up to 382 bytes on the management virtual lane.

C18-5: Switches shall support one of the MTU sizes specified in Table 19 Packet Size on page 168 across all ports on the switch.

C18-6: With the exception of packets arriving on the management virtual lane, switches shall be capable of forwarding packets of size from the minimum valid packet up to the supported MTU plus 126 bytes.

Table 19 Packet Size on page 168 specifies a choice of MTU that may be supported by IBA devices. Switch implementations support one of the specified MTU sizes for the entire switch. Switches are capable of forwarding packets whose size varies up to the maximum size indicated in the table for the implemented MTU size plus an additional 126 bytes.

Link Physicals:

IBA specifies various physical layer options. Switches may implement any of these options on any port and there is no requirement that all ports of a switch implement nor operate with the same physical options. Switches conform to the detailed requirements for physical layer support as specified in Chapter 6: Physical Layer Interface on page 137.

18.2.2 INITIALIZATION

C18-7: Upon power-up, a switch shall be initialized to the following state:

- All initialization of attributes as required in Chapter 14: Subnet Management on page 641.
- Physical and link layers shall be reset.
- All virtual lane queues shall be cleared.
- P_Key enforcement, if implemented, shall be disabled for all ports.
- The NeighborMTU component of each PortInfo attribute shall be initialized to indicate 256 byte MTU as specified in 14.2.5.6 PortInfo on page 665.

Note that a switch contains many tables, some of which are optional. These include the forwarding table, the SL to VL mapping table, the multicast forwarding table, P_Key tables, etc. There is no requirement for a switch to initialize any of these tables; the subnet manager is responsible for appropriate initialization.

18.2.3 CONFIGURATION

Switches are configured via a subnet manager. Switches support the required subnet management operations and may support the optional subnet management operations specified in Chapter 14: Subnet Management on page 641.

18.2.4 PACKET RELAY REQUIREMENTS

The primary function of IBA switches is the relay of packets between links. This section specifies the requirements for supporting this function. This section assumes normal operation; required operation under error conditions is specified in section 18.2.5 Error Handling on page 860.

To simplify the explanation of switch requirements, this section is divided into several architectural functions. This division does not imply a particular implementation; it is done solely to enhance the organization of the specification.

18.2.4.1 SWITCH PORTS

C18-8: Each port on an IBA Switch except port 0 shall comply with the physical layer requirements specified in Chapter 6: Physical Layer Interface on page 137.

C18-9: Each port on an IBA Switch shall comply with the link layer requirements specified in Chapter 7: Link Layer on page 141 of this specification.

C18-10: Port number 0 shall be reserved for the forwarding of packets to and from the switch's Subnet Management Interface and General Services Interface.

C18-10.a1: A switch may support up to 254 physical ports. For a switch supporting N physical ports, the ports shall be numbered from 1 to N.

C18-11: Port number 0 shall comply with the requirements of Chapter 9: Transport Layer on page 203 related to unreliable datagram service.

o18-8: Port 0 shall adhere to all IBA switch port requirements specified in this chapter with the exception that it may deviate from these requirements in any combination of the following ways:

- Port 0 is not required to be physically instantiated.
- Port 0 is not required to implement the IB physical layer electrical, optical, or mechanical requirements.
- Port 0 is not required to implement IB link level flow control.

C18-12: Port 0 shall assume an LMC value of 0.

C18-13: A set of the LMC component of the PortInfo attribute referencing port 0 shall be ignored.

C18-14: All get responses of the PortInfo attribute for port 0 of a switch (including a get response initiated in response to a set operation) shall include a value of 0 for the LMC component.

Port 0 is assigned a LID similar to that of channel adapters; however, unlike channel adapters, this port does not support multipathing and an LMC value cannot be assigned. The LID is assigned using the LID component of the PortInfo attribute. Refer to 14.2.5.6 PortInfo on page 665 for details on these requirements.

18.2.4.2 RECEIVER QUEUING

The receiver queueing function receives packets from the link layer defined in Chapter 7: Link Layer on page 141.

C18-15: The virtual lane into which an individual packet is queued shall be the one corresponding to the VL field in the packet's Local Route Header.

C18-16: If the FilterRawInbound component of the receiving port's PortInfo Attribute is set to one, then the switch shall discard all packets received on that port in which the LNH field of the LRH contains binary 00 or binary 01 (i.e. raw packets).

C18-17: Switches shall not discard packets in lieu of implementation of the link level flow control as specified in section 7.9 Flow Control on page 182.

18.2.4.2.1 INBOUND P_KEY ENFORCEMENT

The implementation of the inbound P_Key enforcement service in switches is optional. This section specifies the requirements of the service if implemented.

Inbound P_Key verification is enabled and disabled for each port individually based on the PatrtitionEnforcementInbound component of the PortInfo attribute.

o18-9: If a switch provides the inbound P_Key enforcement service and the PartitionEnforcementInbound component of the PortInfo Attribute is set to zero, then the inbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o18-10: If a switch provides the inbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o18-11: If a switch provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associated with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o18-12: If a switch provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between one and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o18-13: If a switch provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 674.

o18-14: If a switch provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH does not match one of the entries in the receiving port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

For the purpose of inbound P_Key enforcement, a P_Key matches an entry in the P_Key table if and only if it is not the invalid P_Key one of the following conditions are true:

- The P_Key membership bit in the packet is full and there is an entry in the P_Key table that equals all 16 bits of the P_Key
- The P_Key membership bit in the packet is limited and there is an entry in the P_Key table whose 15 bits exclusive of the membership bit equal those bits in the P_Key.

o18-15: If a switch provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 11.
- IPVer field of the GRH does not contain 6.

o18-16: If a switch provides the inbound P_Key enforcement service and if the PartitionEnforcementInbound component of the PortInfo Attribute is set to one, then any packet that is too short to contain a BTH and that the LNH field contains binary 11 shall be discarded or shall be forwarded with the EBP delimiter appended and with the inverse of the valid VCRC.

Raw packets, i.e. packets in which the LNH field of the LRH contains binary 00 or binary 01, are not subject to P_Key enforcement and are not discarded nor truncated by this mechanism.

18.2.4.3 Packet Relay

Packet relay refers to the operation of transferring a packet from the virtual lane on the inbound port to the virtual lane on a outbound port.

C18-18: A switch shall relay each unicast packet from the data virtual lane(s) in which it was received to the output port indicated by the unicast forwarding table entry corresponding to the packet’s DLID field.

A switch performs this relay function regardless of the state of the destination port. In certain states, the destination port will discard the packet. This is described in detail in 7.2 Link States on page 142.

C18-19: Each packet received on virtual lane 15 in switches that implement independent buffering for virtual lane 15 on each port shall be relayed to the virtual lane 15 on the output port indicated by the unicast forwarding table entry corresponding to the packet's DLID field.

C18-20: If a packet is relayed to the same port on which it was received, it shall be discarded.

(Note: Directed route packets are permitted to be transmitted from the port from which they were received. This does not violate the above requirement since the packet is actually relayed from the received port to port 0, the SMA; then it is received from port 0 and transmitted out the original port).

C18-21: No packet contents shall be modified by the switch except as required by this specification.

This chapter specifies various conditions under which a packet may or must be truncated in length. These conditions do not imply that the PktLen or PayLen fields may be modified.

C18-22: Packets received on ports other than port 0 with a DLID equal to the permissive address shall be forwarded to port 0.

o18-17: If port 0, the SMI, or GSI of a switch does not contain sufficient free buffering to receive the packet, the packet may be discarded.

A special address, the permissive address (see 4.1 Terminology And Concepts on page 116) is defined by IB to permit the subnet manager to communicate with the SMI without knowledge of the LID assigned to the SMI. Packets with the permissive address received on ports other than 0 are always forwarded to port 0.

C18-23: Packets with the permissive address received on port 0 (i.e. generated by the SMI) shall be forwarded to the port specified by the SMI.

The mechanism for the SMI to specify the port is not defined by IB and may vary by implementation.

o18-18: Switches that support more than one virtual lane in addition to the management virtual lane (virtual lane 15), shall set the value of the VL field in the local route header as defined in section 7.6.6 VL Mapping Within a Subnet on page 159.

o18-19: Switches that support one virtual lane in addition to the management virtual lane (virtual lane 15), may implement VL Mapping as defined in section 7.6.6 VL Mapping Within a Subnet on page 159.

o18-20: Switches that support more than one virtual lane in addition to the management virtual lane (virtual lane 15), shall relay packets to the VL of the output port as defined in section 7.6.6 VL Mapping Within a Subnet on page 159 if the corresponding VL is implemented on the output port.

o18-21: Switches that support more than one virtual lane in addition to the management virtual lane (virtual lane 15), shall discard packets if the output VL as defined in section 7.6.6 VL Mapping Within a Subnet on page 159 is not implemented on the output port.

o18-22: Switches that support only one virtual lane in addition to the management virtual lane (virtual lane 15) shall not modify the VL field.

C18-24: Switches that support only one virtual lane in addition to the management virtual lane (virtual lane 15) shall relay packets to the VL of the output port indicated by the VL field in the LRH.

For switches that support only one data virtual lane, the link layer will discard all packets that do not contain either 0 or 15 in the VL field, therefore, there is no need for the relay function to modify the VL field in this case.

C18-25: Except for virtual lane 15, if the virtual lane on the outbound port does not contain sufficient space for the packet to be relayed, then the packet shall remain in the virtual lane on the inbound port until sufficient space is available or until the switch lifetime limit mechanism permits the discard of the packet.

C18-26: If the relay function is unable to relay packet from an inbound port to an outbound port due to lack of sufficient space in the outbound VL, the relay function shall continue to relay packets from other virtual lanes destined for virtual lanes on outbound ports with sufficient space.

o18-23: In switches that implement independent buffering on each port for virtual lane 15, if when relaying virtual lane 15 packets the virtual lane on the output port does not contain sufficient space for the packet to be relayed, then the packet may be discarded.

C18-27: Packets shall be transmitted on a given port and SL in the same order as they were received from a given port except that ordering between unicast and multicast packets is not required.

o18-24: The relay function may, but is not required to, relay packets in the inbound portion of virtual lanes that are behind packets that are blocked due to insufficient space in the outbound portion of virtual lanes.

The method of arbitration when multiple inbound VLs have packets destined for the same outbound VL is left to the implementor, but the arbitration should service all inbound ports fairly.

C18-28: The forwarding table shall be configured in one of two ways, linear or random, as defined in section 18.2.4.3.1 Linear Forwarding Table Requirements on page 854 and 18.2.4.3.2 Random Forwarding Table Requirements on page 855.

C18-29: Switches shall conform to the requirements in section 18.2.4.3.3 Required Multicast Relay on page 856.

o18-25: Switches may implement the requirements in section 18.2.4.3.4 Optional Multicast Relay on page 857.

C18-30: A switch that does not implement the optional multicast relay shall set the MulticastFDBCap component of the SwitchInfo attribute to zero.

18.2.4.3.1 LINEAR FORWARDING TABLE REQUIREMENTS

This section describes the requirements related to the linear forwarding table. The linear forwarding table provides a simple map from LID to destination port. Conceptually, the table itself contains only destination ports; the LID acts as an index into the table from which the packet's destination address is obtained.

C18-31: In switches that implement the linear forwarding table, the linear forwarding table shall contain a port entry for each LID starting from zero and incrementing by one up to the size of the forwarding table.

C18-32: In switches that support the linear forwarding table, the size of the linear forwarding table shall be advertised in the LinearFDBCap component of the SwitchInfo attribute.

C18-33: In switches that support the linear forwarding table, the RandomFDBCap component of the SwitchInfo attribute shall be set to zero.

C18-34: In switches that implement the linear forwarding table, the linear forwarding table shall be programmable using the LinearForwardingTable attribute as described in 14.2.5.10 LinearForwardingTable on page 676.

Note that forwarding to the SMI/GSI is enabled by programming the corresponding entries in the forwarding table to port 0. Setting the LID component of the PortInfo attribute does not automatically load this value in the forwarding table.

C18-35: A switch that implements a linear forwarding table shall support the SM programmable LinearFDBTop component of the SwitchInfo attribute as described in 14.2.5.4 SwitchInfo on page 663.

C18-36: Switches that implement linear forwarding tables shall discard all unicast packets that meet any of the following conditions:

- the packet's DLID value is greater than the value of LinearFDBTop and is not the permissive address
- the packet's DLID is outside the range supported by the linear forwarding table and is not the permissive address
- the port number in the forwarding table corresponding to the packet's DLID is set to a port that does not exist.

18.2.4.3.2 RANDOM FORWARDING TABLE REQUIREMENTS

This section describes the requirements related to the random forwarding table. Conceptually, the random forwarding table acts as a "content addressable memory"; it is loaded with both LIDs and destination ports. The table is "addressed" by a packet's LID and the corresponding destination port is returned. A switch implementation can limit the number of LIDs that correspond to a given port to as few as one. This enables the implementation of a "leaf" switch, i.e., a switch that supports only the connection of CA's to all ports but one. Such a switch requires a very small forwarding table (one LID per port). Such limitations are neither mandated nor prohibited by this specification.

C18-37: In switches that implement the random forwarding table, the random forwarding table shall provide for the storage of a set of unicast LID/LMC pairs and corresponding destination port entries.

C18-38: Switches that implement the random forwarding table shall maintain a DefaultPort value which shall be programmable via the DefaultPort component of the SwitchInfo attribute (see 14.2.5.4 SwitchInfo on page 663 for additional detail).

C18-39: Packets that arrive on ports other than the port indicated by DefaultPort with a unicast DLID field that does not match an entry in the random forwarding table and is not equal to the permissive address shall be forwarded to the port indicated by DefaultPort.

C18-40: If the DefaultPort value is a port that does not exist then packets that would otherwise be forwarded to this port shall be discarded.

C18-41: Packets that arrive on the port indicated by DefaultPort with a unicast DLID field that is not the permissive address and does not match an entry in the random forwarding table shall be discarded.

Matching an entry in the table means that the packet's DLID matches the LID in the table excluding the LMC least significant bits.

C18-42: Switches that implement the random forwarding table shall advertise the size of the table, i.e. the number of LID/LMC pairs that it may contain, in the RandomFDBCap component of the SwitchInfo attribute.

C18-43: Switches that implement the random forwarding table shall set the LinearFDBCap component of the SwitchInfo attribute to zero.

o18-26: Switches that implement a random forwarding table may limit the number of LID/LMC pairs that can be assigned to a given port.

C18-44: If a switch that implements the random forwarding table limits the number of LID/LMC pairs that can be assigned to a given port, then it shall set the LIDsPerPort component of the SwitchInfo component to the number of LIDs that is supported per port.

C18-45: If a switch that implements the random forwarding table does not impose such limitation on the number of LID/LMC pairs that can be assigned to a given port, it shall set the value of the LIDsPerPort component the same as the RandomFDBCap component.

C18-46: In switches that support the random forwarding table, the random forwarding table shall support exactly one LID/LMC entry.

The LIDsPerPort component does not apply to port 0.

18.2.4.3.3 REQUIRED MULTICAST RELAY

C18-47: All switches shall maintain values for a default primary multicast port and a default non-primary multicast port.

All switches maintain values for default primary multicast port and a default non-primary multicast port regardless of whether the switch supports multicast forwarding and regardless of the type of unicast forwarding table implemented.

C18-48: Switches shall allow the SM to set the values of the default primary multicast port and a default non-primary multicast port using the DefaultMulticastPrimaryPort and DefaultMulticastNotPrimaryPort components of the SwitchInfo attribute.

C18-49: All multicast packets that are received on ports other than the default multicast primary port shall be forwarded to the default multicast primary port if any of the following conditions are true:

- The switch does not implement a multicast forwarding table.
- The switch implements a multicast forwarding table and the multicast DLID in the packet is outside the range of the multicast forwarding table.
- The switch implements a multicast forwarding table and the entry in the forwarding table corresponding to the packet's DLID is zero.

C18-50: All multicast packets that are received on the default multicast primary port shall be forwarded to the default multicast non-primary port if any of the following conditions are true:

- The switch does not implement a multicast forwarding table.
- The switch implements a multicast forwarding table and the multicast DLID in the packet is outside the range of the multicast forwarding table.
- The switch implements a multicast forwarding table and the entry in the forwarding table corresponding to the packet's DLID is zero.

C18-51: If either the default multicast primary port or default multicast non-primary port is set to a port that does not exist then multicast packets that would otherwise be forwarded to the corresponding port shall be discarded.

18.2.4.3.4 OPTIONAL MULTICAST RELAY

This section describes the requirements for the optional replication of multicast packets.

o18-27: The replication of packets as part of multicast relay is optional.

o18-28: Switches that support multicast packet replication shall implement a multicast forwarding table that contains a port entry for each multicast LID starting from 0xc000 and sequentially incrementing to include the total number of multicast entries supported.

o18-29: In switches that support multicast packet replication, the number of multicast entries supported in the multicast forwarding table shall be at least one and no greater than 16383.

o18-30: In switches that support multicast packet replication, the number of multicast entries supported in the multicast forwarding table shall be advertised in the MulticastFDBCap component of the SwitchInfo attribute.

o18-31: In switches that support multicast packet replication, if the DLID of a packet is a multicast LID, then the switch shall relay the packet to the set of ports, excluding the port on which the packet was received, indicated by the multicast forwarding table entry corresponding to the packet's DLID field.

o18-32: In switches that support multicast packet replication, the virtual lane field shall be updated in each replicated packet in the same manner as for unicast packets.

18.2.4.4 TRANSMITTER QUEUING

Relayed packets are queued in the outbound portion of virtual lanes.

18.2.4.4.1 OUTBOUND P_KEY ENFORCEMENT

The implementation of the outbound P_Key enforcement service in switches is optional. This section specifies the requirements of the service if implemented.

Outbound P_Key verification shall be enabled and disabled for each port individually based on the PatrtitionEnforcementOutbound component of the PortInfo attribute.

o18-33: If a switch provides the outbound P_Key enforcement service and the PartitionEnfocementOutbound component of the PortInfo Attribute is set to zero, then the outbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o18-34: If a switch provides the outbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o18-35: If a switch provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associated with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o18-36: If a switch provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between one and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o18-37: If a switch provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 674.

o18-38: If a switch provides the outbound P_Key enforcement service and if the PartitionEnfocementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 on that port shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH does not match an entry in the transmitting port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

For the purpose of outbound P_Key enforcement, a P_Key matches an entry in the P_Key table if and only if the P_Key is not the invalid P_Key and one of the following conditions are true:

- The P_Key membership in the packet bit is limited and there is an entry in the P_Key table whose membership bit is full and whose remaining 15 bits equal those of the P_Key
- The P_Key membership bit in the packet is full and there is an entry in the P_Key table whose 15 bits exclusive of the membership bit equal those bits in the P_Key.

o18-39: If a switch provides the outbound P_Key enforcement service and if the PartitionEnforcementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 of that port shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 11.
- IPVer field of the GRH does not contain 6.

o18-40: If a switch provides the outbound P_Key enforcement service and if the PartitionEnforcementOutbound component of the PortInfo Attribute is set to one, then any packet that is too short to contain a BTH and that the LNH field contains binary 11 shall be discarded or shall be forwarded with the EBP delimiter appended and with the inverse of the valid VCRC.

Raw packets, i.e. packets in which the LNH field of the LRH contains binary 00 or binary 01, are not subject to P_Key enforcement and are not discarded nor truncated by this mechanism

18.2.4.5 PACKET TRANSMISSION

C18-52: If the FilterRawOutbound component of the transmitting port's PortInfo attribute is set to one, then the switch shall discard all packets to be transmitted on that port in which the LNH field of the LRH contains binary 00 or binary 01 (i.e. raw packets).

C18-53: Each packet shall be transmitted with a valid VCRC field computed as specified in section 7.8.2 Variant CRC (VCRC) - 2 Bytes on page 170, unless required otherwise in this chapter or in chapter Chapter 7: Link Layer on page 141.

C18-54: Each packet shall be transmitted with an egp character appended unless required otherwise in this chapter.

C18-55: Switches shall support the requirements specified in 7.6.9 VL Arbitration and Prioritization on page 161.

18.2.5 ERROR HANDLING

This section specifies required operation under error conditions. Like the previous section, this section is divided into several architectural functions. This division does not imply a particular implementation; it is done solely to enhance the organization of the specification.

18.2.5.1 SWITCH PORTS

C18-56: Each port except port 0 on an IBA Switch shall comply with the physical layer error requirements specified in Chapter 6: Physical Layer Interface on page 137.

C18-57: Each port except port 0 on an IBA Switch shall comply with the link layer error requirements specified in Chapter 7: Link Layer on page 141 of this specification.

18.2.5.2 RECEIVER QUEUING

There are no additional receiver queuing error handling requirements.

18.2.5.3 PACKET RELAY

There are no additional packet relay error handling requirements.

18.2.5.4 TRANSMITTER QUEUEING

The transmitter packet discard is based on, among other things, two time values: Switch Lifetime Limit (SLL) and Head of Queue Lifetime Limit (HLL).

SLL is defined as 4.096us * 2^{LV} if $0 \le LV \le 19$, +5% / -55%. LV is the LifeTimeValue component of the SwitchInfo attribute. If LV > 19, then SLL is to be interpreted as infinite.

HLL is defined as 4.096us * 2^{HL} if $0 \le HL \le 19$, +5% / -55%. HL is the HOQLife component of the PortInfo attribute. If HL > 19, then HLL is to be interpreted as infinite.

C18-58: The transmitter queueing function shall discard any packet that meets any of the following conditions:

- The packet has been at the head of the Virtual Lane (i.e. the position to be transmitted next), and has not begun transmission within HLL.
- The packet is queued to a VL that is in the VL stalled state. If VL-StallCount sequential packets are discarded from a given VL due to exceeding the HLL requirement above, the VL shall enter the VL stalled state. A VL shall leave the VL stalled state 8 * HLL after entering it. VLStallCount component is provided in the PortInfo attribute.
- The size of the packet as indicated by the PktLen field exceeds the MTU supported by the neighbor device as indicated by the NeighborMTU component of the PortInfo attribute.

C18-59: If a switch by virtue of its implementation cannot guarantee that any packet entering it will be transmitted within 2.5 ms, measured first bit in to first bit out and assuming flow control credit is continuously available, then it shall discard any packet that has not begun transmission within SLL measured from the time the first bit was received by the switch.

o18-41: If a switch by virtue of its implementation can guarantee that any packet entering it will be transmitted within 2.5 ms, measured first bit in to first bit out and assuming flow control credit is continuously available, then it may discard any packet that has not begun transmission within SLL measured from the time the first bit was received by the switch.

18.2.5.5 PACKET TRANSMISSION

C18-60: Each packet to be transmitted that is truncated in length as permitted or specified by any condition in this chapter be corrupted as specified in 7.3 Packet Receiver States on page 146.

18.2.6 SUBNET MANAGEMENT AGENT REQUIREMENTS

C18-61: Switches shall support a Subnet Management Interface (SMI) as specified in Chapter 14: Subnet Management on page 641.

C18-62: Switches shall support a General Services Interface (GSI) as specified by Chapter 16: General Services on page 748.

There are various mandatory and optional requirements of these interfaces that are specified in the respective chapters.

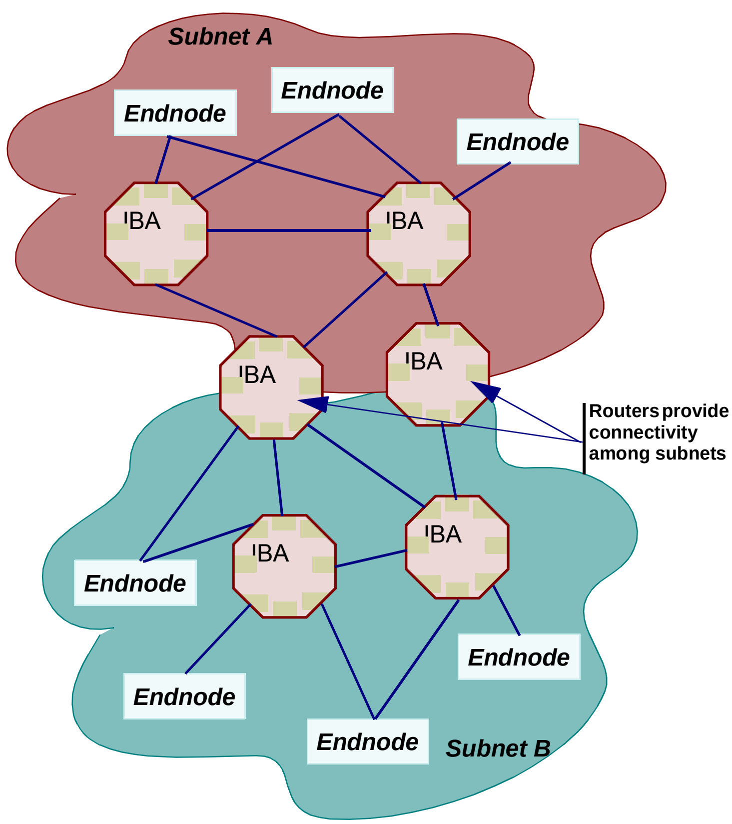
C18-63: Switches shall implement P_Key checking on the GSI as specified in section 10.9.8 Partition Enforcement on Management Queue Pairs on page 460.

CHAPTER 19: ROUTERS

19.1 OVERVIEW

IBA Routers are IBA packet relay devices, that operate at the network layer of the IBA addressing hierarchy to interconnect multiple locally addressed subnets. As the top level in the hierarchy, IBA Routers rely on global identifiers (GIDs).

Figure 200 Reference of Routers Connecting Subnets



IBA Router usage is meant to satisfy:

1) Scalability
2) Local address space reuse
3) Containment of failures and topology changes
4) Confinement of fabric management scope to subnets

In fulfilling these objectives, IBA Routers also allow the IBA semantics, and QoS characteristics to be extended across IBA subnets.

IBA Routers are required to support unicast routing and may support multicast routing. The specification of the routing forwarding mechanisms in this chapter is presently limited to unicast routing.

IBA Routers use destination based routing, where every destination port within the global fabric is assigned one or more unique Global Identifiers (GID). From the point of view of a Router, a GID represents either an end-node port or another router's port on a directly attached subnet. A GID does not necessarily represent a path through the fabric, as the Router is allowed to spread traffic over several paths based on other packet header criteria.

A Router is visible to IBA nodes on the directly attached subnets, and it is transparent to nodes on any remote subnet. The Subnet Manager address resolution function makes local routers visible to endnodes; endnodes in turn use this information when addressing packets to a local router LID on their way to a remote destination. Routers on the same subnet are also visible to each other, both for the purpose of implementing a routing protocol, and also when routing packets through other routers as the next hop. Finally, routers on a subnet are also visible to Subnet Managers on their respective directly attached subnets.

Each Router port must support a Subnet Management Interface (SMI) (see 13.5.1.1 Processing Subnet Management Packets (SMPs) on page 632) and a General Services Interface (GSI) (see 13.5.1.2 Processing General Services Management Packets (GMPs) on page 632). There are various mandatory and optional requirements of this interface that are specified in the management chapter. Subnet Managers assign LIDs to IBA Router ports and provide a service to find a path to other endnodes or routers. The Router Management section of the Management chapter defines the attributes of the interactions between Subnet Managers and Routers.

19.2 DETAILED FUNCTIONAL REQUIREMENTS

The present IBA Router specification does not cover the routing protocol nor the messages exchanged between routers. Future revisions of this chapter will complete such control functions.

19.2.1 ATTRIBUTES

IBA Routers reside at the boundaries between subnets, and are configured separately per port by different Subnet Managers and at different times. Subnet managers supply IBA Routers with LIDs/LMCs (for each port separately), and additional path information like SL to VL mappings and MTU values.

Unicast Routing Table:

C19-1: A router shall implement a unicast routing table with at least as many entries as the number of router ports.

IBA Routers have routing tables for their active routes. These tables are hierarchical and include explicit endnode routes (e.g. last hop to endnode), prefix routes (aggregate route for entire subnet), and possibly default routes (routes for unknown prefixes). The size of the unicast routing table is implementation dependent.

Virtual lanes:

C19-2: Routers shall implement the subnet management virtual lane (also referred to as virtual lane 15).

C19-3: Router ports shall implement data virtual lanes as specified in 7.6 Virtual Lanes Mechanisms on page 153, in addition to the subnet management virtual lane, numbered virtual lane 15.

C19-4: Virtual lane 15 shall be implemented independently for each router port.

C19-5: Routers shall not route any VL15 packets between router ports.

SL to VL mapping:

C19-6: Routers that implement more than one data virtual lane shall implement the SL to VL mapping function specified in this chapter.

o19-1: Routers that implement one data virtual lane may implement the SL to VL mapping function specified in this chapter.

Per port SL to VL mapping is required on Routers that support more than one virtual lane in addition to virtual lane 15. It is optional on Routers that support only one virtual lane in addition to virtual lane 15.

Tclass to SL mapping:

C19-7: Routers shall preserve the Tclass value when routing.

SL values may be replaced when routing into a different subnet. The Tclass value is preserved, as it represents the Class of Service with end-to-end IBA scope. The Tclass to SL mapping function is not defined by the present specification revision.

P_Key Enforcement:

o19-2: Routers may implement the Inbound P_Key Enforcement Service specified in this chapter.

o19-3: Routers may implement the Outbound P_Key Enforcement Service specified in this chapter.

Routers may enforce partitions on ingress to and/or egress from the Router.

Maximum Transfer Unit (MTU) size:

C19-8: Each router port shall independently support one of the MTU sizes specified in Table 19 Packet Size on page 168.

C19-9: Routers shall be capable of routing packets of size from the minimum valid packet size up to the supported MTU of the intervening ports plus 126 bytes.

Table 19 Packet Size on page 168 specifies a choice of MTU that may be supported by IBA devices. Router implementations independently support one of the specified MTU sizes for each port. Packets exceeding the MTU size of the participating links may be discarded or truncated. Each port provides sufficient buffering for each data VL to advertise credit for at least one packet with MTU payload.

Link Physicals:

IBA specifies various physical layer options. Routers may implement any of these options on any port and there is no requirement that all ports of a Router implement nor operate with the same physical options. Routers shall conform to the detailed requirements for physical layer support as specified in Chapter 6: Physical Layer Interface on page 137.

End-to-end data integrity:

Although IBA routers modify some packet headers during routing, none of these headers affects the value of the ICRC, and IBA Routers shall preserve the original ICRC rather than recomputing its value locally.

19.2.2 INITIALIZATION

C19-10: Upon power-up, a router shall be initialized to the following state:

- All initialization defined in Chapter 13: Management Model on page 595 and applicable to routers.
- Physical and link layers reset.
- All virtual lane queues shall be cleared.
- Other routing table entries and all queues (and any other table type structures) shall be cleared.

19.2.3 Configuration

Routing tables - Entries may be derived from any combination of externally configured routes and autonomously computed routes. In particular, routers rely on the SM database for routes to endnodes on directly attached subnets.

SL mapping tables - SL to VL mapping tables exist at every router port.

Tclass mapping - Any necessary configuration of the Tclass end-to-end class of service role in determining the local SL value is configured into the router.

19.2.4 Packet Relay Model

The logical abstraction for IBA packet routing is that of packet by packet routing and is given by:

if

i) ((BASE DLID == router port BASE LID) AND

ii) (LRH:Next Header == GRH) AND

iii) (Destination GID <> router GID) AND

iv) (Destination GID matches entry in route table) AND

v) (VCRC OK) AND (Hop count > 1))

then {

i) Replace DLID with value from routing table

ii) Replace SLID with LID of output port

iii) Replace SL, considering Tclass (among other possible criteria)

iv) Map SL to VL using per output port table

v) Decrement Hop count

vi) Recompute VCRC, preserve ICRC

}

Note: Routers may also check ICRC, or just rely on the end-to-protection of endnodes checking ICRC.

The above abstraction, combined with the Network layer Addressing model, dictates a longest match against the destination GID. Implementations may exploit the addressing model to relax this function to a combination of a 64-bit longest match for prefix type entries, and either a 64-bit or 128-bit fixed length match for explicit routes, depending on the uniqueness scope of the lowest 64 bits of the GID.

19.2.4.1 PATH SELECTION

A Router may support multiple paths to a given DGID. These may include paths via the same next-hop and/or different next-hops as well as different paths within the subnet (LMC based) to a given GID.

An IBA Router may actively use multiple paths with equal or different costs, as long as it does not affect ordering by separating packets of a given session.To allow IBA Routers to have different degrees of sophistication in determining what packets may be separated, a session is used in a deliberately vague way.

The baseline assumption is that endpoints will use identical GRH:FlowLabel values for sequences of packets whose relative ordering is important, therefore a possible session representation at the router would be the **(DGID, SGID, TClass, SL, FlowLabel)** tuple. A router may use other attributes to select a path but once selected that path will continue to be used for subsequent packets unless a management event dictates a new path.

19.2.4.2 ROUTER PORTS

A router with N physical ports, associates PORTINFO attributes to its physical ports based on the Port Number Attribute Modifier value, ranging between 1 and N.

C19-11: Each port on an IBA Router shall comply with the physical layer requirements specified in Chapter 6: Physical Layer Interface on page 137.

C19-12: Each port on an IBA Router shall comply with the link layer requirements specified in Chapter 7: Link Layer on page 141 of this specification.

C19-13: Each port on an IBA Router shall implement the Subnet Management PORTINFO Attribute specified in 14.2.5.6 PortInfo on page 665.

C19-14: Each port on an IBA Router shall have at least one GID assigned to it.

19.2.4.3 Receiver Queuing

The receiver queueing function receives packets from the link layer defined in Chapter 7: Link Layer on page 141.

C19-15: The virtual lane into which an individual packet is queued shall be the virtual lane whose virtual lane number matches the VL field in the packet's Local Route Header.

C19-16: If the FilterRawInbound component of the receiving port's PortInfo attribute is set to one, then the Router shall discard all packets received on that port in which the LNH field of the LRH contains a binary 00 or binary 01 (i.e. raw packets).

19.2.4.3.1 Inbound P_Key Enforcement

The implementation of the inbound P_Key enforcement in Routers is optional. This section defines its requirements if implemented.

Inbound P_Key verification shall be enabled and disabled for each port individually based on the PartitionEnforcementInbound component of the PortInfo attribute.

o19-4: If a router provides the inbound P_Key enforcement service and the PartitionEnfocementInbound component of the PortInfo Attribute is set to zero, then the inbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o19-5: If a router provides the inbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o19-6: If a router provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associated with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o19-7: If a router provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between 1 and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o19-8: If a router provides the inbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 674.

o19-9: If a router provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH is not contained in the receiving port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

o19-10: If a router provides the inbound P_Key enforcement service and if the PartitionEnfocementInbound component of the PortInfo Attribute is set to one, then any packet received on a virtual lane other than 15 shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 11.
- IPVer field of the GRH does not contain 6.

Raw packets are not subject to P_Key enforcement and shall not be discarded nor truncated by this mechanism.

19.2.4.4 PACKET RELAY

Packet relay refers to the operation of transferring a packet from the virtual lane on the inbound port to the virtual lane on a outbound port. The output virtual lane selection is specified later in this section. The relay function is performed regardless of the state of the destination port. In certain port states the destination port will discard the packet.

C19-17: A router shall relay each unicast packet from the virtual lane zero through fourteen (if implemented) in which it was received to the output port indicated by the routing table entry corresponding to the packet's DGID field.

C19-18: Packets received on virtual lane 15 shall not be relayed to output ports.

A packet may be relayed to the same port on which it was received, this is necessary to support some routing scenarios like endnodes using one out of several routers on the subnet as a default router.

o19-11: Routers that support more than one virtual lane, in addition to virtual lane 15, shall set the value of the VL field in the local route header by first considering the GRH Tclass field to derive a SL value for the subnet attached to the output port and then using the SL to VL mapping scheme as defined in section 7.6.6 VL Mapping Within a Subnet on page 159.

C19-19: Routers shall always recognize and map a Tclass value of 0 to a best effort SL.

C19-20: Each packet relayed from an inbound port shall be placed on the virtual lane of the outbound port specified by the SL to VL mapping. If the new value of VL does not correspond to a configured VL on the outbound port, the packet shall be discarded. Also, packets received with a VL not configured for the port shall be discarded.

C19-21: If the virtual lane on the outbound port does not contain sufficient space for the packet to be relayed, then the packet shall remain in the virtual lane on the inbound port until sufficient space is available or until the router lifetime limit mechanism permits the discard of the packet.

C19-22: If the relay function is unable to relay packet from an inbound port to an outbound port due to lack of sufficient space in the outbound VL, the relay function shall continue to relay packets from other virtual lanes destined for virtual lanes on outbound ports with sufficient space.

C19-23: Packets shall be transmitted on a given port and SL in the same order as they were received from a given port.

o19-12: The relay function may, but is not required to, relay packets in the inbound portion of virtual lanes that are behind packets that are blocked due to insufficient space in the outbound portion of virtual lanes.

The method of arbitration when multiple inbound VLs have packets destined for the same outbound VL is left to the implementor, but the arbitration should service all inbound ports fairly.

C19-24: Routers shall not continuously assert backpressure (i.e. fail to grant link credits). Regardless of what congestion policy an IBA router associates to its relay function, routers shall not cause deadlock in the fabric.

C19-25: Packets whose Hop count is less than 2 shall be discarded.

C19-26: The Hop count of every relayed packet is decremented by one.

19.2.4.5 Transmitter Queuing

Relayed packets shall be queued in the outbound portion of virtual lanes.

19.2.4.5.1 Outbound P_Key Enforcement

The implementation of the outbound P_Key enforcement service in routers is optional. This section specifies the requirements of the service if implemented.

Outbound P_Key verification shall be enabled and disabled for each port individually based on the PatrtitionEnforcementOutbound component of the PortInfo attribute.

o19-13: If a router provides the outbound P_Key enforcement service and the PartitionEnfocementOutbound component of the PortInfo Attribute is set to zero, then the outbound P_key enforcement service shall be disabled for packets received on the corresponding port.

o19-14: If a router provides the outbound P_Key enforcement service, it shall maintain a separate list of P_Keys associated with each port.

o19-15: If a router provides both the inbound P_Key enforcement service and the outbound P_Key enforcement service, then the list of P_Keys associate with each port shall be the same list for both the inbound P_Key enforcement service and the outbound P_Key enforcement service.

o19-16: If a router provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be capable of containing between 1 and 65535 P_Keys, inclusive (the exact number is left as an implementation parameter).

o19-17: If a router provides the outbound P_Key enforcement service, the P_Key table associated with each port shall be programmable using the P_KeyTable attribute defined in 14.2.5.7 P_KeyTable on page 674.

o19-18: If a router provides the outbound P_Key enforcement service and if the PartitionEnfocementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 on that port shall either be discarded or truncated such that it contains no data past the BTH if the value in the P_Key field in the BTH is not contained in the transmitting port's P_Key list and either of the following conditions are true:

- LNH field in the LRH contains binary 11 and IPVer field in the GRH contains 6.
- LNH field in the LRH contains binary 10.

o19-19: If a router provides the outbound P_Key enforcement service and if the PartitionEnfocementOutbound component of the PortInfo Attribute is set to one, then any packet to be transmitted on a virtual lane other than 15 of that port shall either be discarded or truncated in length such that it contains no more than 64 bytes if all of the following conditions are true:

- LNH field of the LRH contains binary 01 or binary 11.
- IPVer field of the GRH does not contain 6.

Raw packets, i.e. packets in which the LNH field of the LRH contains binary 00 or 01, are not subject to P_Key enforcement and are not discarded nor truncated by this mechanism.

19.2.4.6 PACKET TRANSMISSION

C19-27: If the FilterRawOutbound component of the transmitting port's PortInfo attribute is set to one, then the router shall discard all packets to be transmitted on that port in which the LNH field of the LRH contains a binary 00 or binary 01 (i.e. raw packets).

C19-28: Routers shall perform SL to VL mapping as defined in 7.6.6 VL Mapping Within a Subnet on page 159. This mapping is based on the outbound SL to be used for the packet.

C19-29: Packet shall be transmitted with a valid VCRC field computed as specified in section 7.8.2 Variant CRC (VCRC) - 2 Bytes on page 170, unless required otherwise in this chapter.

C19-30: Each packet shall be transmitted with an EGP character appended unless required otherwise in this chapter.

Routers shall support the requirements specified in 7.6.9 VL Arbitration and Prioritization on page 161.

19.2.5 ERROR HANDLING

This section specifies required operation under error conditions for each of the conceptual functions.

19.2.5.1 ROUTER PORTS ERRORS

C19-31: Each port on an IBA router shall comply with the physical layer error requirements specified in Chapter 6: Physical Layer Interface on page 137.

C19-32: Each port on an IBA router shall comply with the link layer error requirements specified in Chapter 7: Link Layer on page 141 of this specification.

19.2.5.2 RECEIVER QUEUING ERRORS

The receiver queueing function may discard any packet that meets any of the following conditions:

- There is insufficient space in the virtual lane to receive a packet of the size indicated in the PktLen field in the local route header.
- The size of the packet indicated by the PktLen field in the local route header indicates that the packet exceeds the MTU size supported by the Router port.

The receiver queueing function may discard any packet if its transmission has not been initiated and if the packet meets any of the following conditions:

- There is insufficient space in the virtual lane to receive the packet.
- The packet exceeds the MTU size supported by the output port.
- A VCRC error was detected on reception.
- The length of the received packet was different from that indicated by LRH:PktLen.
- The packet has a framing error.
- The packet was received with an EBP delimiter appended.
- The length of the packet was too short to contain a LRH, GRH, and a VCRC.

C19-33: Any packet that exceeds the MTU size supported by the output port and that is not discarded shall be truncated to any size that meets the MTU size limitation of the port.

19.2.5.3 PACKET RELAY ERRORS

C19-34: Packets with no GRH, or with a GRH version not supported by the Router shall be discarded.

19.2.5.4 TRANSMITTER QUEUEING ERRORS

C19-35: Routers shall implement the Packet Lifetime limits and Head of Queue Lifetime Limit mechanisms defined for IBA Switches in 18.2.5.4 Transmitter Queueing on page 860.

The Packet Lifetime limit is determined from the LifeTimeValue component of the RouterInfo attribute using the same formula as the Switch Lifetime Limit. The Head of Queue Lifetime Limit is determined from the HOQLife component of the PortInfo attribute using the same formula as its switch counterpart.

The above mentioned limits on packet lifetime inside IBA routers and switches are meant to help drain packets from the IBA fabric before they can present a hazard to the IBA transport layer finite sequence number space. These limits are not defined as congestion management mechanisms, and should not be reached in normal circumstances, even in congestion scenarios.

19.2.5.5 PACKET TRANSMISSION ERRORS

C19-36: Each packet to be transmitted that was received with an error indicated by the link layer shall be transmitted with an EBP character appended and the VCRC field shall contain the one's complement of the valid VCRC.

o19-20: Each packet to be transmitted that was received with an error indicated by the link layer may be truncated in length.

C19-37: Each packet to be transmitted that is truncated in length as permitted or specified by any condition in this chapter be corrupted as specified in 7.3 Packet Receiver States on page 146.

19.2.6 SUBNET MANAGEMENT AGENT REQUIREMENTS

C19-38: Each router port shall implement a Subnet Management Interface (SMI) as specified in [Chapter 14: Subnet Management on page 641.

C19-39: Routers shall support a General Services Interface (GSI) as specified in Chapter 16: General Services on page 748.

The General Services Interface is used, for example, for GID to LID address resolution.

CHAPTER 20: VOLUME 1 COMPLIANCE SUMMARY

20.1 COMPLIANCE DEFINITION

This chapter specifies the Compliance Categories that are approved for labeling various products that contain InfiniBand content. This will allow vendors to label their products and claim InfiniBand compliance without creating confusion in the marketplace. This chapter addresses compliance to the feature set defined by Volume 1 of the InfiniBand Specification.

20.1.1 PRODUCT APPLICATION

Each product that has InfiniBand content **may** claim InfiniBand Compliance to one or more of the Categories defined in the Compliance Summaries of the InfiniBand Specification. A product **shall not** simply claim "InfiniBand Compliant".

Each claim of compliance **shall** be a list of one or more valid InfiniBand Compliance Categories from Volume 1 or Volume 2. It's appropriate for some products to include Compliance Categories from both Volumes 1 and 2.

The valid Volume 1 Compliance Categories are defined below.

Those for Volume 2 are defined in *InfiniBand Architecture Specification, Volume 2* Chapter "Volume 2 Compliance Summary".

20.2 VOLUME 1 COMPLIANCE CATEGORIES

Volume 1 Compliance Categories refer to the functionality of each entity defined in Volume 1. Table 258 on page 876 lists all valid Volume 1 Compliance Categories along with their full names.

Because optional functionality may be associated with a given Compliance *Category*, zero or more Compliance *Qualifiers* may be associated with that Category. Table 258 lists all valid Qualifiers under each Category. Qualifiers shown in ***bold italics*** indicate functionality that is actually non-optional for that specific category, but those Qualifiers may still appear in some of the Compliance Statements listed under that Category.

Table 259 on page 878 lists Volume 1's complete set of Qualifiers along with their full names. Section 20.2.1 discusses Qualifiers in more depth.

Each Category has a dedicated section in this chapter that contains, among other things, a complete reference list of Volume 1 compliance statements that directly apply to that category. Table 258 provides a reference to each section, including a hypertext link with on-line versions of the spec.

Table 258 Volume 1 Compliance Categories

Category	Full Name	Valid Qualifiers	Reference
HCA-CI	Host Channel Adapter - Channel Interface	VLs, ***RC***, ***UC***, RD, RawD, APM, UDMcast, RawDMcast, ***RDMA***, Atomics, P_Key traps, P_Key counters, Notice, Trap	Section 20.3 on page 880
TCA	Target Channel Adapter	VLs, RC, UC, RD, RawD, APM, UDMcast, RawDMcast, RDMA, Atomics, P_Key traps, P_Key counters, Notice, Trap	Section 20.4 on page 892
SW	Switch	VLs, UDMcast, P_Key SRE, P_Key SRE_In, P_Key SRE_Out, Notice, Trap	Section 20.5 on page 900
RTR	Router	VLs, RawD, UDMcast, RawDMcast, P_Key SRE, Notice, Trap	Section 20.6 on page 904
SM	Subnet Manager	Trap	Section 20.7 on page 907
SA	Subnet Administration	UDMcast, Trap, SAOPT	Section 20.8 on page 908
CM	Communication Manager	APM	Section 20.9 on page 909
PFM	Performance Manager	Trap	Section 20.10 on page 909
VM	Vendor-Defined Manager	Trap	Section 20.11 on page 910
OMA	Optional Management Agent	Trap, AMA, DMA, SNMP, VMA	Section 20.12 on page 910

20.2.1 VOLUME 1 COMPLIANCE QUALIFIERS

Compliance Qualifiers indicate which compliance statements apply only if a product supports an optional feature or specified combination of optional features.

Some compliance statements apply to multiple Compliance Categories, and thus appear in the Compliance Statement List under each applicable Category. Some of these “shared” compliance statements include Qualifiers associated with functionality that is optional in some Categories and mandatory in others. In each Category where the functionality is mandatory, the associated Qualifier is shown in ***bold italics*** for that Category’s “Valid Qualifier’s” entry in Table 258.

20.2.1.1 CLAIMING SUPPORT FOR OPTIONAL FEATURES

C20-1: If a product claims to support a given optional feature, the product **must** comply with **all** compliance statements that apply to that optional feature.

For example, an HCA-CI that claims to support Reliable Datagram Service must comply with all statements under the HCA-CI Compliance Statement List that apply, given the RD Qualifier.

A product **shall not** include in its list of supported *optional* features any features that are in fact *mandatory* for the Category the product claims compliance to. Qualifiers for these mandatory features are shown in ***bold italics*** in Table 258. For example, Reliable Connection Service is mandatory for HCA-CI, so RC must not be included in an HCA-CI’s list of supported optional features even though the product must still meet all RC requirements.

A product may claim support for multiple optional features, in which case the product must comply with all compliance statements that apply to the particular set of optional features claimed by the product, noting that some compliance statements apply only for specific combinations of qualifiers.

Table 259 lists and describes the Volume 1 Compliance Qualifiers that a product can claim compliance to. To abbreviate the optional support, one or more Qualifiers can be listed after the Category in braces. For example, a Target Channel Adapter that supports Reliable Datagram Service and also Automatic Path Migration can be abbreviated with TCA{RD,APM}

Table 259 Volume 1 Compliance Qualifiers

Qualifier	Description
VLs	Port Supporting More than One Data VL
RC	Reliable Connection Service
UC	Unreliable Connection Service
RD	Reliable Datagram Service
RawD	Raw Datagram Service
APM	Automatic Path Migration
UDMcast	Unreliable Datagram Multicast
RawDMcast	Raw Datagram Multicast
RDMA	Remote Direct Memory Access
Atomics	Atomic Operations
P_Key SRE	P_Key Enforcement by Switches or Routers
P_Key SRE_In	Inbound P_Key Enforcement by Switches or Routers
P_Key SRE_Out	Outbound P_Key Enforcement by Switches or Routers
P_Key traps	Trap Generation for P_Key Violations
P_Key counters	Counters for P_Key Violations
Notice	Standard Format & Queue for Data About Events
Trap	Asynchronous Event Notification
SAOPT	Subnet Administration Bulk Update Facilities
AMA	Application-specific Management Agent
DMA	Device Management Agent
SNMP	SNMP Tunneling Agent
VMA	Vendor-specific Management Agent

20.2.1.2 COMPLIANCE STATEMENTS WITH MULTIPLE QUALIFIERS

Some compliance statements contain combinations of Qualifiers, and apply only if the specified combination is true. For example, a compliance statement beginning with “RD and Atomics:” applies only if *both* RD and Atomics are supported. If a compliance statement begins with “RD or Atomics:”, the statement shall apply if *either* RD or Atomics is supported.

20.2.2 COMPLIANCE STATEMENT LISTS

Within each Compliance Category section is a list of the compliance statements that apply to that particular category. Here is a sample list entry:

- **o9-16:** RD: PSN Insertion for Reliable Svc Pkts. . . . . Page 231

20.2.2.1 HYPERTEXT LINKS

Online versions of this specification have hypertext links present before each of the lines in the Compliance Statement lists. These links are indicated by the "●" at the beginning of the line and will lead to the actual statement in the body of the specification that contains the details for each of the compliance entries.

Each Compliance Statement List entry also contains the page number for use with hard-copy versions of the specification.

20.2.2.2 COMPLIANCE STATEMENT LABELS

All formal compliance statements throughout the specification are labeled so they can be uniquely identified. Each label begins with either a "C" or an "o", indicating whether the compliance statement applies in all cases with respect to its category or whether the compliance statement is qualified with respect to optional features. The "o" is uncapitalized to make it more easily distinguishable from the "C" in Compliance Statement Lists.

The next portion of the label is the number of the chapter in which the formal compliance statement appears. The final portion of the label is a compliance statement number, which starts with "1" in each chapter. "C" and "o" compliance statements are numbered independently.

20.2.2.3 COMPLIANCE STATEMENT TITLES

Each line within a Compliance Statement List contains a brief title for the respective compliance statement. Because of the limited space and lack of context, each title is only intended to convey the topic of the compliance statement, and not necessarily convey its actual requirements.

Compliance statements that apply only to optional functionality is indicated by the presence of one or more Qualifiers at the beginning of the title, followed by a colon. For example, the above sample Compliance Statement Title contains the "RD" qualifier.

20.2.3 COMMON REQUIREMENTS

Some Compliance Categories share common requirements, such as those that apply to all ports. To avoid unnecessary duplication, certain common requirement sets have been collected and referenced by the appropriate Compliance Categories instead of replicating those lists of requirements under each separate Category.

20.3 HCA-CI COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of HCA-CI, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant HCA-CI shall meet all Section 20.13 Common Port Requirements on page 911 and all Section 20.14 Common MAD Requirements on page 912.

Some compliance statements in the HCA-CI Category contain requirements that apply to both mandatory and optional features. For instance, some compliance statements mention both QPs and EE Contexts, though EE Contexts are relevant only if RD Service is supported. In such cases, the requirements on an optional feature apply only if the product claims to support the optional feature.

20.4 TCA COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of TCA, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant TCA shall meet all Section 20.13 Common Port Requirements on page 911 and all Section 20.14 Common MAD Requirements on page 912.

20.5 SWITCH COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Switch, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Switch shall meet all Section 20.13 Common Port Requirements on page 911 and all Section 20.14 Common MAD Requirements on page 912.

20.6 ROUTER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Router, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support. In addition, a compliant Router shall meet all Section 20.13 Common Port Requirements on page 911 and all Section 20.14 Common MAD Requirements on page 912.

20.7 SUBNET MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Subnet Manager, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

20.8 Subnet Administration Compliance Category

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Subnet Administration, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

20.9 COMMUNICATION MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Communication Manager, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

Though a number of optional CM-specific features exist, no CM-unique qualifiers have been defined since the optional CM-specific features are fully described within the CM Chapter, and most of the chapter's optional compliance statements would require unique qualifiers.

20.10 PERFORMANCE MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Performance Manager, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

20.11 VENDOR-DEFINED MANAGER COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Vendor-Defined Manager, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

20.12 OPTIONAL MANAGEMENT AGENT COMPLIANCE CATEGORY

In order to claim compliance to the InfiniBand Volume 1 specification to the Compliance Category of Optional Management Agent, a product shall meet all requirements specified in this section, except for those statements preceded by Qualifiers that the product does not support.

20.13 Common Port Requirements

Multiple Compliance Categories share common Port Requirements. To avoid unnecessary duplication, Port Requirements are collected here and referenced by the appropriate Compliance Categories.

20.14 COMMON MAD REQUIREMENTS

Multiple Compliance Categories share common Management Datagram Requirements. To avoid unnecessary duplication, Management Datagram Requirements are collected here and referenced by the appropriate Compliance Categories.